EmXJ : 유연한 임베딩을 위한 XML 처리기 구성 프레임워크

정 원 호[†] · 강 미 연[††]

요 약

유무선 인터넷의 발달로, 휴대폰, PDA, 홈패드, 스마트폰, 핸드헬드 PC 등, 다양한 형태의 자원제약을 가지는 개인용 혹은 상업용 임베디드 시스템들이 속속 등장하고 있으며, 이들로 임베딩되는 소프트웨어도, 과거 이들이 지녔던 주요 특성인 경직성(fixedness)보다도, 이제는 오히려 유연성(flexibility)을 요구하고 있다. 즉, 자원 제약 특성이 서로 다른 장치들로 유연하게 임베딩될 수 있는 특성을 요구하고 있다. 웹 상의 정보 표현을 위한 표준으로 자리잡은 확장성 표기 언어인 XML을 위한 처리기는 각종 인터넷 단말에서의 자료 브라우징을 위해 필수적으로 임베딩 되어야할 소프트웨어이다. 본 논문에서는 서로 다른 기능과 자원 제약 특성을 가진 장치들에 따라 유연성 있는 임베딩을 위한 XML 처리기 구성 프레임워크인, EmXJ가 설계, 구현되며, 그 특성이 기존의 XML 처리기와 비교 분석된다.

EmXJ : A Framework of Configurable XML Processor for Flexible Embedding

Won-Ho Chung[†] · Mi-Yeon Kang[††]

ABSTRACT

With the rapid development of wired or wireless Internet, various kinds of resource constrained mobile devices, such as cellular phone, PDA, homepad, smart phone, handheld PC, and so on, have been emerging into personal or commercial usages. Most software to be embedded into those devices has been forced to have the characteristic of flexibility rather than the fixedness which was an inherent property of embedded systems. It means that recent technologies require the flexible embedding into the variety of resource constrained mobile devices. A document processor for XML which has been positioned as a standard mark-up language for information representation on the Web, is one of the essential software to be embedded into those devices for browsing the information. In this paper, a framework for configurable XML processor called EmXJ is designed and implemented for flexible embedding into various types of resource constrained mobile devices, and its advantages are compared to conventional XML processors.

키워드 : EmXJ, Primitive-모듈, Macro-모듈, 유연 임베딩(Flexible Embedding), XML 처리기(XML processor)

1. 서 론

인터넷의 급속한 보급에 따라, 전자 문서의 표현 및 교환에 대한 상용화의 필요성이 증대되어, W3C에서는 구조화된 전자문서의 표현 및 교환을 위하여, 기존의 HTML과 SGML의 장점들을 포괄하는 확장성 표기 언어인 XML에 대한 표준안을 제시하였다[1]. XML을 이용한 효율적이고 통일적인 문서 정보의 관리가 이루어질 수 있게 되면서, XML의 응용 범위는 전자상거래를 거점으로 데이터베이스, 멀티미디어 표현 등 관련 분야로 급속하게 넓혀지고 있다. XML의 응용 분야가 전자 비즈니스 전 분야로 확대되면서, XML 기반의 다양한 응용 제품 및 솔루션의 개발에 대한 필요성이 증대되고 있다. 특히, 유·무선 인터넷 기술의 연동 및 통합은 각종 비즈니스 분야에 모바일 환경의 구축을 유도하고 있으며, 휴대폰, PDA, 홈패드, 스마트폰, 핸드헬드 PC 등, 서로 다른 유형의 기능 및 자원 제약을 가지는 모바일 임베디드 시스템들의 상용화를 가속화시키고 있다. 웹 상의 정보 표현을 위한 표준으로 자리잡은 확장성 표기 언어인 XML을 위한 처리기는, XML 문서의 브라우징을 위해서, 이러한 다양한 유형의 장치에 필수적으로 임베딩 되어야할 소프트웨어 중 하나이다. XML 처리기는 파서를 가장 기본적인 컴포넌트로 가지고 있으며, C 혹은 Java 언어를 기반으로 개발되었으나, 플랫폼 독립 특성을 가지기 위해서 최근에는 주로 Java를 기반으로 개발되고 있다[2-7]. 파서 엔진을 축으로 하는 XML 처리기도 이제는 임베디드 소프트웨어로서, 유연한 임베딩을 위해 구성적 특성을 지니도록 요구받고 있다. 즉, 단순 기능과 자원 사용의 최소화를 지향하는 저급 장치는 최소한의 기능을 가지면서 고

※ 본 연구는 2002년도 산업자원부 중기거점과제 "디지털 가전형 POST-PC 플랫폼 기술개발사업" 지원으로 이루어졌음.
† 정 회 원 : 덕성여자대학교 컴퓨터과학부 교수
†† 준 회 원 : ICANTEK(주) 기술연구소
논문접수 : 2002년 9월 28일, 심사완료 : 2002년 11월 26일

속 처리가 가능한 XML 처리기를 요구하고 있으며, 반면에, 다양한 기능과 활용 자원의 제약이 적은 고급 장치는 여러 기능을 가진 중량급의 XML 처리기를 필요로 할 수도 있는 것이다.

임베디드 소프트웨어의 주요 특징은 경량화(light weight)와 실시간(real-time)성이라 할 수 있다. 경량화는 기능의 경직화 및 단순화를 근간으로 하며, 실시간화는 하드웨어 가속기의 사용, 알고리즘의 효율화 및 기능의 단순화 등을 통해 이루어진다. 그러므로, 기능의 단순화는 소프트웨어의 경량화 및 실시간화를 위한 한 가지 방법이라 할 수 있다. 특정 하드웨어에 가장 적절하고 효율적으로 고정되는 경직성(fixedness)은 사용된 하드웨어 자원의 변화를 허용하지 않는다. 이는 제한된 자원내에서 특수 목적의 작업을 실시간으로 처리하기 위해서 취할 수밖에 없었던 설계 개념이라고 볼 수 있으며, 과거에는 그러한 실시간 장치들이 기능적 단순성을 가지고 있었으며, 극히 제한된 분야에서만 사용되었다[8]. 그러나, 최근 들어, 유·무선 인터넷 기술의 발달로, 저급 PDA로부터 고급 핸드헬드 PC까지 매우 다양한 수준의 기능과 규모의 하드웨어 자원을 가지는 포스트 PC급 장치들이 속속 개발되어 상품화되고 있는 실정에 비추어 볼 때, 이제 임베디드 소프트웨어도 유연성이 뛰어나도록 설계 개념이 바뀌어야 한다는 시각이 나타나고 있다[8]. 운영체제는 Linux처럼, 공개 소스 기반의 임베디드 운영체제를 지향하고 있으며[9], HAVi와 Jini처럼, 프레임워크 기반의 임베디드 소프트웨어의 개발을 위한 연구도 활발히 진행중에 있다[8, 10]. 무선 인터넷 기반으로 많이 이용되는 Java의 경우, 속도의 저하 문제를 극복하기 위해, 실시간 특성을 추가하는 것과 단말로의 효율적 임베딩을 위한 연구가 활발히 진행 중이며[11, 12], 각종 단말 장치들이 웹 서버로서의 기능을 수행하는 응용이 적지 않아, 최근에는 임베디드 웹 서버에 관한 연구도 활발히 진행중에 있다[13-15].

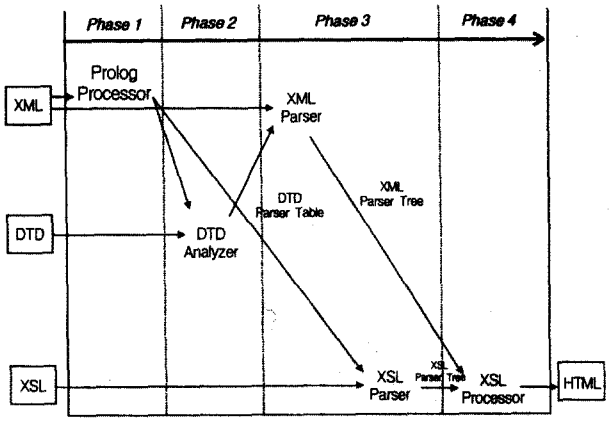
본 논문에서는 서로 다른 기능과 규모의 메모리 자원 제약을 가지는 장치들로의 유연한 임베딩을 위해 선택적 기능 및 메모리 공간을 차지하는 Java 기반의 XML 처리기 구성 방법이 제안되며, 이를 기반으로, EmXJ라고 하는 XML 처리기 구성 프레임워크가 설계, 구현된다. 메모리 사용 제약이 심한 저급의 휴대폰, PDA 등과 같은 장치들을 위한 단순 기능의 경량급 XML 처리기의 구성과 자원 제약이 별로 없는 다양한 기능을 가진 핸드헬드 PC같은 고급 단말을 위한 XML 처리기의 구성이 EmXJ를 기반으로 유연하게 구축될 수 있다. EmXJ는 자원 규모 제약이 서로 다른 다양한 정보 단말 하드웨어에 체계적으로 임베딩 할 수 있도록 함을 목적으로 하면서도, 그 과정이 복잡하고 난해하지 않다는 장점을 가지고 있다. 이는 소프트웨어의 기술적 이전 측면에서 매우 중요한 부분이라 할 수 있으며, 지속적인 시스템의 용이한 업그레이드를 위해서도 중요한 측면이다.

본 논문의 구성은 다음과 같다. 2장에서는 DOM 기반 및 SAX 기반의 XML 처리기 동작 및 구조에 관한 소개와 임베딩시 고려되어야 할 사항이 기술되며, 3장에서는, 본 연구에서 설계·개발된 XML 처리기 구성 프레임워크인 EmXJ에 관한 특징과 구조, 그리고 자원의 규모에 맞는 기능 및 크기를 가지는 XML 처리기의 구성 개념 등이 기술된다. 4장에서는 Java 기반의 다른 XML 처리기들과의 소요 메모리 공간 및 기능 등의 비교 분석을 통해 EmXJ의 장점이 기술된다. 5장에서는 EmXJ를 사용하여 구성된 SAX 기반 XML 처리기와 DOM 기반의 XML 처리기의 간단한 응용 예가 기술되며, 마지막으로 6장에서 결론 및 향후 연구가 기술된다.

2. XML 처리기의 개요

XML 문서는 일반적으로 3 부분으로 구성된다. 먼저, 문서의 도입부로, 문서 전체의 특징, 예를 들면, XML 버전, DTD의 유무 및 위치 등을 기술하는 프롤로그 부분, 문서에서 사용되는 개체들(element, attribute, entity 등)의 정의 및 그들 사이의 관계 등을 기술하는 DTD(Document Type Declaration) 부분, 그리고 DTD가 존재할 경우, DTD에서 정의된 개체 및 서술 규칙에 맞게 기술된 내용 부분으로 구성된다[1]. XML 처리기는, DTD와 XML 문서를 입력으로 하여, DTD의 구문 검사 및 정확성 검사를 하고 문서의 내용 개체들을 트리 형태로 변환하여 메모리 상에 구성함으로써 응용 프로그램으로 하여금 문서 내용의 접근 또는 변경을 가능하게 한다. 생성된 트리는 문서에 포함된 모든 요소 정보를 저장하고 있는데, 문서를 스캐닝하면서 정형화(wellformedness) 및 정확성(validity) 여부를 검증하면서 구축된다. XML 문서는 문서 내용에 대한 구조적 정보만을 가지고 있으며, 문서 내용의 스타일에 관한 정보는 가지고 있지 않기 때문에, XML 문서를 특정 출력 장치에 시각적으로 표현하기 위해서는 문서가 표시되는 방식에 대한 규정을 담고 있는 확장성 양식 언어인 XSL을 사용한다. 이러한 XSL은 W3C에서 제시한 XSL 형식객체(XSL-fo)를 사용하는 방법과, XML 문서의 HTML 문서로의 변환을 통해 표시하는 XSLT 방식이 있다[17, 18]. 전자의 경우에는 XSL-fo를 지원하는 브라우저가 필요하나, 후자의 경우는 XML 문서를 HTML로 변환하는 과정을 거치므로, HTML 브라우저를 그대로 사용할 수 있다는 장점을 가진다. XML 처리기의 일반적인 동작과정을 보여주고 있는 것이 (그림 1)이다.

XML 처리기의 사용은 주로 다음 2가지 측면에서 이루어진다. 하나는 MIE 혹은 넷스케이프와 같은 XML 문서 처리를 지원하는 상용화된 소프트웨어를 사용하는 경우로, 처리기에 대한 동작 및 기능은 전적으로 사용 소프트웨어에서의 사양에 의존하는 경우이다. 사용이 용이하고, 문서 접근을 위

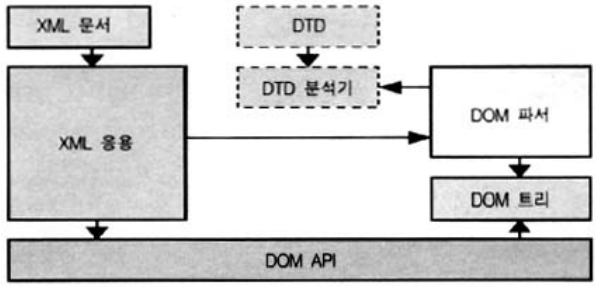


(그림 1) XML 처리기의 동작 과정

위한 간단한 동작은 가능하나, 그 응용 범위가 극히 한정되게 된다는 단점을 가진다. 다른 하나는 XML 처리기를 핵심 컴포넌트로 하여 특정 XML 솔루션을 개발하는 경우이다. 이때, XML 처리기가 제공하는 각종 API들을 이용하여 해당 응용 소프트웨어를 개발한다. 본 연구에서 추구하는 XML 처리기의 용도는 후자에 두고 있다. W3C에서는 XML 응용 개발을 용이하게 하도록, XML 처리기가 제공해야 하는 API들을 표준화하였다. 이러한 API에는 2 종류가 있으며, 문서객체 모델(Document Object Model, DOM)을 기반으로 하는 API와 이벤트 구동 방식을 기반으로 하는 SAX(Simple API for XML) API가 그것들이다.

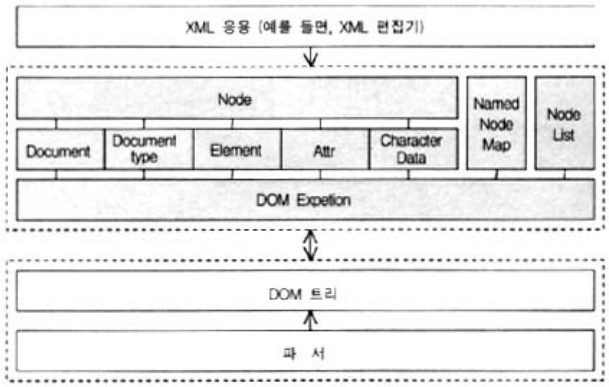
2.1 DOM 기반의 XML 처리기의 구성 및 동작

XML 문서의 구조와 내용은 element, attribute, text 등으로 구성된 트리 구조의 계층화된 객체들의 집합으로 표현 될 수 있으며, 트리상의 객체들의 접근에 관한 인터페이스를 표준으로 정해 놓은 것이 DOM이며, 이 트리를 DOM 트리라고 한다. DOM 기반 XML 처리기(이하, DOM 처리기)는 문서를 파싱하면서, 문서의 논리적 구조에 관한 정보와 내용을 객체로서 메모리 상에 상주시키고 필요한 정보를 원활하게 액세스할 수 있도록 하는 API를 제공하고 있다. 그러므로, DOM 처리기는 문서로부터 DOM 트리를 생성하며, 응용은 DOM에서 제공된 API를 이용하여 DOM 트리를 탐색하거나, 노드 값으로의 접근 및 변경, 노드의 추가와 삭제를 할 수 있으며, 더 나아가 DOM 트리를 다시 XML 문서로 변환할 수도 있다. 그러므로, DOM 처리기는 문서 구조 및 내용 전체가 메모리 상에 저장되어 있어, 문서의 일부를 여러 번 읽어야 하는 경우나 문서의 구조가 충실히 보존되어야 하는 경우를 비롯하여, 문서를 빈번히 수정해야 하는 응용의 경우에 이용하면 적합하다. 간단한 응용 예로 XML 편집기를 들 수 있다. DOM 처리기와 응용과의 동작 관계를 간략하게 보여주고 있는 것이 (그림 2)이다.



(그림 2) DOM 처리기의 동작 블록 다이어그램

최근에 완성 발표된 DOM Level 2는 (그림 3)에서 보여준 바와 같은 구조를 가지며, Core, Events, Style, Traversal and Range, 그리고 Views 등 5개의 모듈로 구성되어 있으나, 모든 경우를 지원하기 위해서는 처리기의 크기가 방대해지므로 대부분 일부만을 지원하는 것이 일반적이다[4-6].

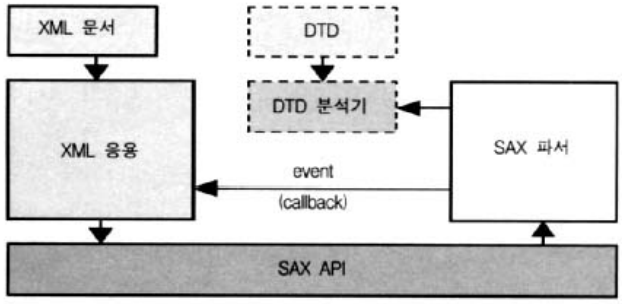


(그림 3) DOM 기반 XML 처리기의 구조

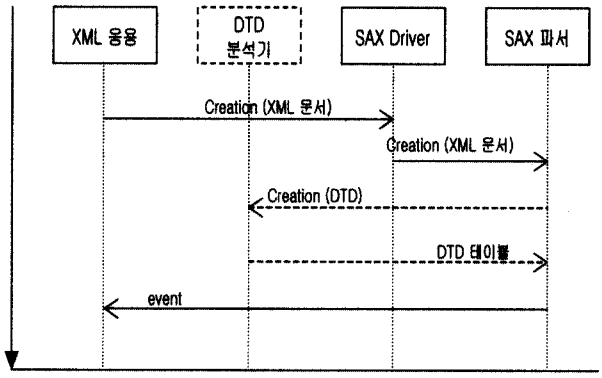
2.2 SAX 기반의 XML 처리기의 구성 및 동작

SAX 기반 XML 처리기(이하, SAX 처리기)는 문서를 처음부터 끝까지 순차적으로 처리한다. 수행 해야할 기능들을 최소화시켜 경량급 고속처리를 추구하는 SAX 처리기는 문서를 읽어가면서 element나 attribute 등의 출현을 알려주는 이벤트 기반의 API를 정의하고 있다. DOM 처리기와는 달리, 전체 문서를 트리구조와 같은 체계적인 형태로 메모리에 저장하지 않고, 문서의 구조를 일련의 이벤트 흐름으로 표현하며, 이러한 이벤트를 만날 때마다, 해당 SAX API를 응용쪽으로 콜백(callback)하게 된다. 그러므로, SAX 처리기를 사용하는 응용에서는 콜백함수 수행을 위해 해당 SAX API를 구현하여야 한다. 문서를 저장하지 않고 순차적으로 처리하는 경우나, 문서가 간단하고 그 구조 자체가 주요 관심사가 아닌 경우에 이용하기 적합하다. 단순히 어떤 element, attribute 등의 정보를 줄 뿐, 문서의 구조적 정보를 유지해 주지 않기 때문에, 문서의 구조를 바탕으로 하는 정보를 요구할 경우, 원하는 정보에 관한 추출은 모두 응용에서 이루어져야 한다. 그러므로, 단순한 기능을 제공하며, 그 외 복잡한 기능은 모두 응

용에서 이루어지도록 하고 있어, SAX 처리기는 DOM 처리기에 비해 기능이 단순하고 경량급이므로 임베디드 XML 처리기로 많이 사용되고 있다[4, 5]. 이러한 경우, 대부분 DTD 관련 기능들은 지원하지 않는데, 그것은 코드 크기가 클 뿐만 아니라 필요로 하는 heap 데이터 공간 또한 크기 때문이다. SAX 처리기와 응용과의 동작 관계와 순서를 간략하게 보여주고 있는 것이 (그림 4)이다.



(a) SAX 처리기의 동작 블록 다이어그램

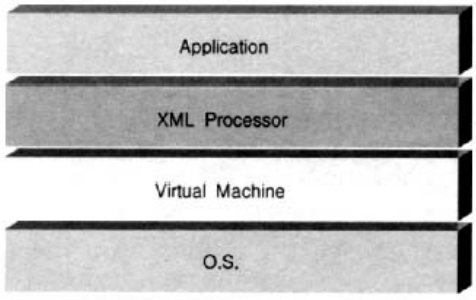


(b) SAX 처리기의 동작 순서를 나타내는 순차 다이어그램

(그림 4) SAX 처리기 동작 및 순서

2.3 임베디드 XML 처리기의 특징

XML 문서 처리를 위한 Java 기반의 임베디드 시스템에서의 소프트웨어 구조는 (그림 5)에서 보여준 바와 같은 계층 구조를 가지고 있다. 임베디드 시스템의 특성 중의 하나인 경량화는 각 계층의 기능 담당 소프트웨어의 경량화를 통해 이루어지며, 이는 기능의 단순화와 경직화를 통해 이루어 질 수 있다. 그러나, PDA, 홈패드, 스마트폰, 핸드헬드 PC 등, 여러 형태의 자원 제약을 가지는 개인용 혹은 상업용 임베디드 시스템들의 등장으로, 이들로 임베딩되는 소프트웨어도, 과거 이들이 지녔던 주요 특성인 단순성, 경직성(fixedness)을 더 이상 고집할 수 없으며, 이제는 오히려 유연성(flexibility)을 요구하고 있다. 임베디드 운영체제는 Linux처럼, 공개 소스 기반의 임베디드 시스템을 지향하고 있으며[9, 19], 정보 가전 시스템은 전용 프레임워크를 기반으로 임베디드 소프트웨어를 개발하고 있다[8, 10]. 무선 인터넷 플랫폼으로 많이 이용되는 Java 가상 기계의 경우에도, 유연한 임베딩을 위한 연구가 활발히 진행중에 있으며, 새로운 개념의 가상기계들이 개발되고 있다[11, 12, 20]. XML 처리기도 예외일 수 없다. 서로 다른 기능 및 자원 제약 특성을 가지는 여러 유형의 장치에 응용과 함께 임베딩될 XML 처리기를 유연하게 구성하기 위해서는, 응용에서 요구하는 기능에 따른 코드 공간의 크기와 heap 데이터 공간의 크기가 임베딩시 주의 깊게 고려되어야 할 것이다.

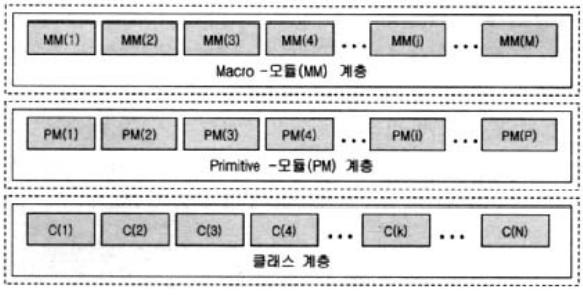


(그림 5) 일반적인 Java 기반 임베디드 시스템에서의 소프트웨어 구성

3. EmXJ의 구조 및 기능

3.1 Primitive-모듈과 Macro-모듈

EmXJ는, 임베딩될 장치에서 요구하는 기능 및 활용 가능 메모리 공간(코드 및 heap 공간)을 고려하여, 응용과 함께 임베딩 될 XML 처리기의 구성을 가능하게 하는 프레임워크로, (그림 6)에 보여준 바와 같이 *클래스* 계층, *Primitive-모듈(PM)* 계층, 그리고 *Macro-모듈(MM)* 계층으로 구성된 계층 구조를 가진다. 최하위 계층인 클래스 계층은 XML 문서 처리를 위해 필요로 하는 각종 클래스들로 구성된다. 그러나, 이들 독자적으로는 XML 처리기를 위한 독립적인 어떤 기능도 이루어지지 않는다. 클래스 계층의 클래스들을 조합하면, XML 처리기에서 필요한 독립적인 하나의 기능을 정의할 수 있는데. 이처럼 클래스들의 집합을 통해 하나의 독립적 기능을 이루면서 더 이상 세분할 필요가 없는 클래스들의 집합을 Primitive-모듈(PM)이라고 정의한다. 예를 들면, 토크나이징, 트리 생성, 문법 조사, 오류 처리 등과 같은 XML 문서 처리를 위한 기본 기능들이, 여기에 해당된다.



(그림 6) EmXJ의 계층 구조

〈표 1〉 PM 계층을 구성하는 Primitive-모듈들

순번	PM's		크기 (KB)	담 당 기 능
1	PMBasicParser	XMLBP	15.8	문서의 tokenizing, syntax check 및 error 처리
2	PMParseTree	XMLPT	5.5	문서에 대한 parse tree 생성 및 탐색
3	PMSAXDriver	SAXDR	4.8	SAX parser 구동 모듈로, element와 CharacterData에 대한 event 발생
4	PMSAXAttr	SAXAT	3.9	Attribute list에 대한 event 발생
5	PMSAXErrHandler	SAXER	0.2	Error event 발생시 응용 레벨에서 오류 처리를 위한 SAX API들의 집합
6	PMSAXReader	SXRDA	3.0	Element와 CharacterData에 대한 event 발생을 나타내는 SAX API들의 집합
7	PMSAXAttributes	SXATA	0.4	Attribute list에 대한 event 발생을 나타내는 SAX API들의집합
8	PMSAXException	SXEXA	0.8	Parsing 중 발생하는 exception event 발생을 나타내는 SAX API들의 집합
9	PMProlog	XMLPG	16.7	문서의 Prolog 부분을 처리하는 DOM API들의 집합 및 구현
10	PMElement	XMLEL	14.0	문서의 element 및 character data를 처리하는 DOM API들의 집합 및 구현
11	PMAttributes	XMLAT	3.4	문서의 attribute list를 처리하는 DOM API들의 집합 및 구현
12	PMException	XMLEX	0.9	Parsing 중 발생하는 exception을 처리하는 DOM API들의 집합 및 구현
13	PMDTDAnalyzer	XMLDT	28.5	DTD를 분석하여 element 및 attribute에 관한 자료구조(DTD table) 구성
14	PMXSLTProc	XSLTP	38.5	문서의 viewing을 위한 XSLT 스타일 문서를 파싱하여 parse tree 구성 및 이를 바탕으로 XML 문서를 HTML 문서로 변환

현재, EmXJ의 PM 계층은 <표 1>에서 보여준 바와 같이, 14개의 모듈들로 구성되어 있다. 예를 들어, PMBasicParser (XMLBP)는 주어진 XML 문서에서 사용된 토큰들을 가려내고, 구문조사를 통하여 오류가 발생하였을 경우, 오류 처리를 담당하는 기능을 가지고 있으며, 3개의 클래스로 구성되어 있다. XML 처리기의 기능을 확장 또는 변경하기 위해, 새로운 모듈 혹은 기존 모듈들을 PM 계층에 추가 혹은 변경할 수 있다. 이는 계속 변경 수정되고 있는 XML 사양에 대해 효율적으로 대처할 수 있는, EmXJ가 지니고 있는, 좋은 특성이라 할 수 있다.

PM 계층의 모듈들을 기반으로 하여, 독립된 XML 처리기로서의 역할이 가능한 모듈을 구성할 수 있는데, 이를 Macro-모듈(MM)이라고 정의한다. 비록, PM 계층의 모듈들로 부터 임의의 Macro - 모듈이 생성 가능하다 할지라도, XML 처리기로서의 독립된 기능이 가능하지 못하면 MM 계층에 속할 수 없다. MM 계층에 속한 Macro - 모듈들은 모두 이러한 개념을 기반으로 구성되어 있다. PM 계층의 모듈들로부터 Macro - 모듈의 구성은 <표 2>에서 보여준 바와 같은 구성(configuration) 테이블을 기반으로 이루어진다.

3.2 구성(Configuration) 테이블

PM 계층에서 정의된 모듈들의 선택적 조합을 기반으로 하여, Macro -모듈의 생성을 보여주는 테이블을 구성 테이블이라고 하며, 현재, EmXJ가 가지는 구성 테이블은 <표 2>에 보여준 바와 같다. MM 계층에 속한 각 Macro - 모듈, MM(k)는 그 자체로 XML 처리기 역할을 할 수 있으나, 그 보유 기능은 MM(k)를 구성하고 있는 Primitive - 모듈들에 따라 각각 다르다. PM 계층에 새로운 모듈의 추가 및 변경이 가능하듯이, 구성 테이블 역시 Primitive - 모듈의 추가와 변경에 따라 새로운 Macro - 모듈의 추가와 변경이 가능하다. <표 2>에서 보아 알 수 있듯이, 현재 EmXJ의 MM 계층은 MM(1)에서 MM(10)까지 10가지의 Macro - 모듈들을 지원하고 있으며, 이들은 크게 3가지 유형의 XML 처리기로 분류될 수 있다. 즉, MM(1)은 하나의 Primitive - 모듈 XMLBP로 구성되며, 기본적인 구문 조사만 가능한 구문 검사기(syntax

〈표 2〉 MM 계층의 Macro-모듈 구성을 위한 구성(configuration) 테이블

PM's	Macro - 모듈, MM(*k*)									
	1	2	3	4	5	6	7	8	9	10
XMLBP	●	●	●	●	●	●	●	●	●	●
XMLPT		●					●	●	●	●
SAXDR			●	●	●	●				
SAXAT					●	●				
SAXER				●		●				
SXRDA			●	●	●	●				
SXATA					●	●				
SXEXA				●		●				
XMLPG							●	●	●	●
XMLEL							●	●	●	●
XMLAT									●	●
XMLEX								●		●
Code Size (KB)	15.8	21.3	23.6	24.6	27.9	28.9	52.0	52.9	55.4	56.3

checker)로 가장 단순한 기능을 가지는 XML 처리기라고 할 수 있다. MM(2)는 XMLBP와 XMLPT 2개의 Primitive - 모듈들로 구성되며, MM(1)의 기능에 트리 생성 및 탐색이 가능한 파서이다. 그러므로, 구문 검사 및 트리 생성 기능을 가진 기본적인 XML 파서라고 할 수 있다. MM(3)부터 MM(6)까지는 구문 검사기인 XMLBP와 SAX 드라이버 모듈인 SAXDR을 공통 구성요소로 가지고 있는 SAX 처리기들이다. SAX 처리기는 MM(3)부터 MM(6)까지 4단계로 분류되며, MM(3)는 가장 단순한 SAX 처리기이며, MM(6)는 EmXJ에서 지원하는 모든 SAX API의 기능을 가지는 SAX 처리기이다. 그리고, MM(7)부터 MM(10)까지는 기본 파서인 MM(2)를 공통 구성요소로 가지고 있는 DOM 처리기들이다. DOM 처리기, 또한 MM(7)부터 MM(10)까지 4 단계로 분류되며, MM(7)은 가장 단순한 DOM 처리기이며, MM(10)은 EmXJ에서 지원하는 모든 DOM API의 기능을 지원하는 DOM 처리기이다.

3.3 Macro-모듈의 기능별 분류

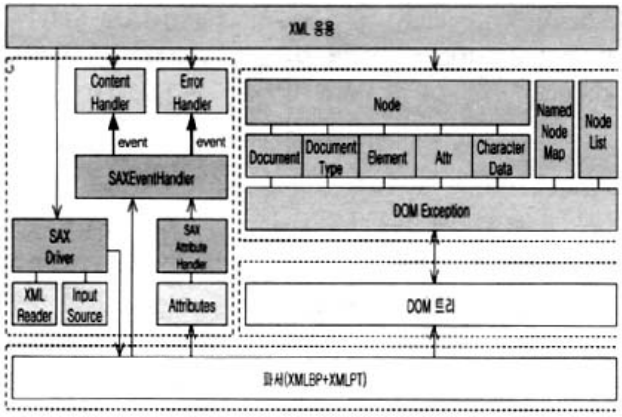
앞에서 언급했듯이, MM 계층을 구성하는 각 Macro - 모듈, MM(k)는 그 자체로서 XML 처리기이다. 가장 최소의 기능을 가지는 MM(1)부터, DOM 처리기 중에서 가장 다양한 기능을 가진 MM(10)까지의 각 MM(k)에 대한 기능 및 코드 크기를 기술하고 있는 것이 <표 3>이다. MM(1)과 MM(2)는 앞에서 언급했듯이, 2가지 유형의 기본 파서이며, MM(3)부터 MM(6)은 보유 기능에 따라 4 단계로 나뉘어진 SAX 처리기들이다. 예를 들어, MM(3)는 MM(1)을 기반으로 하여, XML 문서상의 엘리먼트와 문자데이터에 대한 이벤트를 구동시키는 반면, 속성(attribute) 리스트나 오류에 대한 이벤트 구동 기능에 대한 API를 지원하지 않는, 가장 단순한 SAX 처리기이다. 그러므로, XML 문서가 속성 리스트를 포함하거나, 오류를 포함하고 있어도 이벤트를 구동시키지 않으므로 응용에서는 속성 리스트의 처리나 오류 처리 API를 사용할 수 없으며, 기본 파서가 제공하는 오류 정보에 의존할 수밖에 없다. 반면, MM(6)는 속성 리스트 및 오류에 대한 이벤트 구동 기능의 API를 지원하므로, 속성 리스트를 포함하는 XML 문서를 대상으로 응용을 해야하는 SAX 처리기를 원한다면, MM(6)가 바람직하다고 할 수 있다. MM(7)부터 MM(10)까지는 기능 별로 4 단계로 나뉘어진 DOM 처리기이다. SAX 처리기에서와 마찬가지로, MM(7)은 속성 리스트나 오류 처리 기능 API를 지원하지 않는, 즉 엘리먼트와 문자 데이터 처리 기능만 가지는 기본적인 DOM 처리기이며, MM(10)은 EmXJ에서 지원하는 모든 DOM API 기능을 가지고 있는 DOM 처리기이다. 현재, EmXJ 상에서 구성 가능한 가장 단순한 XML 구문 검사기인 MM(1)의 코드 크기는, <표 3>에서 보아 알 수 있듯이, 15.8KB 이며, 가장 복잡한 DOM 처리기인 MM(10)은 56.3KB이다.

더 나아가서, 구성 테이블 상에서 SAX API와 DOM API를 동시에 지원하는 XML 처리기의 구성도 가능하다. 즉, MM(1)부터 MM(10)까지의 처리기들 중 2개 이상의 처리기의 조합을 통해 원하는 처리기를 구성할 수 있다. 예를 들어, MM(6)와 MM(10)의 조합인, $MM(6) \cup MM(10)$의 조합을 통해 XML 처리기를 구성할 수 있는데, 이는 EmXJ가 지원하는 모든 SAX API와 DOM API를 지원하는 가장 복합적인 XML 처리기라고 할 수 있으며, 코드 크기는 약 90KB 이다.

이와 같이 구성되어, SAX API 및 DOM Core API를 모두 지원하는 XML 처리기에 대한 구조를 보여주고 있는 것이 (그림 7)이다. 그러므로, MM 계층의 MM(k)들을 목적하는 응용에 필요한 기능 및 활용 가능한 메모리 공간을 고려하여 적절하게 조합하면 단순한 기능의 XML 처리기부터

〈표 3〉 각 MM(k)의 기능 및 코드별 분류

MM(*k*)	XML 처리기 유형	기 능	크기 (KB)
1	구문 검사기	문서의 토크나이징, 구문 검사 및 오류 처리	15.8
2	기본 파서	문서를 토크나이징, 구문 검사, 그리고 파스 트리 생성 및 탐색	21.3
3	기능별 SAX 처리기	파서를 구동하여, 엘리먼트와 문자 데이터에 대한 이벤트 발생 및 응용 레벨에서 이벤트 처리를 위한 SAX API 제공	23.6
4		MM(3)의 기능과 오류 이벤트 발생 및 처리, 그리고 파싱 중 발생하는 예외 이벤트에 대한 SAX API와 응용에서 이벤트 처리를 위한 SAX API 제공	24.6
5		MM(3)의 기능 및 속성 리스트에 대한 이벤트 발생, 그리고 응용에서 이벤트 처리를 위한 SAX API 제공	27.9
6		MM(4) 및 MM(5)의 기능	28.9
7	기능별 DOM 처리기	문서의 토크나이징, 구문 검사 및 파스 트리 생성, 그리고 문서의 Prolog 부분과 엘리먼트 및 문자 데이터 처리 및 파스 트리 탐색 DOM API 제공	52.0
8		MM(7)의 기능 및 파싱 중 발생하는 예외 처리를 위한 DOM API 제공	52.9
9		MM(7)의 기능 및 속성 리스트를 처리하는 DOM API 제공	55.4
10		MM(8) 및 MM(9)의 기능	56.3



(그림 7) $MM(6) \cup MM(10)$의 조합을 통해 구성된 XML 처리기의 구조

복합 기능을 가진 XML 처리기까지 다양한 수준의 XML 처리기를 구성할 수 있다.

EmXJ는 저급 휴대폰 혹은 PDA와 같은 장치를 위한 경량급 XML 처리기 부터, 고급 Post-PC급 장치를 위한 중량급 XML 처리기까지를 다르게 구성하여 임베딩 시킬 수 있도록 하고 있다. 예를 들어, 활용 가능 메모리 공간이 매우 작은 저급 PDA에서의 경우, 문서 정보의 탐색이나 변경 등의 고급 기능을 요하지 않는 것이 일반적이다. 즉, 단순한 기능의 경량급 XML 처리기를 필요로 하므로, MM(3) 수준의 SAX 처리기 정도면 충분하다고 할 수 있으며, 반면에, 활용 메모리 공간이 충분하여 고급 기능을 요하는 응용의 경우, MM(7)부터 MM(10)중의 하나로 구성된 DOM 처리기를 구성하거나, 더 원하면 2개 이상의 MM(k)의 조합을 통해 SAX와 DOM을 모두 지원하는 고급 XML 처리기를 구성하여 사용할 수 있다.

일반적으로 잘 알려진 2가지 유형의 XML 처리기로 DTD를 사용하지 않는 정형(wellformed) XML 처리기와 DTD를 사용하는 적합(valid) XML 처리기에 대한 구성을 보여주고 있는 것이 <표 4>이다. DTD를 사용하는 적합 XML 처리기는 정형 XML 처리기에 PM 계층에 속하는 모듈 XMLDT를 추가하여 구성된다. 보통, XML 브라우징은 응용에서 이루어지는 것이 보통인데, 만약에, XML 브라우징 기능을 XML 처리기에 추가하고 싶다면, PM 계층의 또 하나의 모듈인 XSLTP를 추가하여 구성할 수 있다.

4. 소요 메모리 공간 및 기능 비교

4.1 소요 메모리 공간 특성

EmXJ는, 기능적인 측면에서, XML 1.0 사양을 따르고, SAX 2.0 및 DOM 레벨 2 Core를 지원하는 XML 처리기까지 구성할 수 있다. DOM 레벨 2 사양의 나머지 모듈들은 PM 및 MM 계층에 적절한 Primitive - 모듈 및 Macro - 모듈들을 추가함으로써 용이하게 확장될 수 있다.

활용 메모리 공간의 제약을 고려하여 XML 처리기를 구성할 경우, 코드 크기보다 더 주요한 요인으로 고려되어야 할 것은 수행을 위해 필요한 heap 데이터 공간의 크기라고 할 수 있다. 그것은, 3장에서 언급했듯이, EmXJ에서 구성 가능한 XML 처리기의 코드 크기는 약 20KB(MM(1))에서 약 90KB ($MM(6) \cup MM(10)$) 범위에 존재하고 있으며, 물론 이 크기도 기능의 추가 및 확장에 따라 증가할 수 있으나, 그에 따른 heap 공간의 크기 증가는 코드 증가에 비해 훨씬 클 수 있으며, 이러한 heap 공간의 크기는 보통 XML 문서에서 사용된 엘리먼트 혹은 속성의 수에 비례하여 증가하게 된다.

(그림 8)은 EmXJ에서 구성되는 DOM 트리의 각 엘리먼트 노드에 해당하는 구조와 객체를 보여주고 있으며, 그 크기는 약 0.7KB이다. 마찬가지로, 속성 노드에 해당하는 객체와 텍스트 노드에 해당하는 객체의 크기는 각각 약 0.58KB와 0.5 KB를 차지하고 있다. 그러므로, 어떤 XML 문서에서 사용된 엘리먼트의 개수가 E이고 속성의 개수가 A라면, 이를 위해 필요한 heap 공간의 크기는 $(0.70 \cdot E + 0.58 \cdot A)$ KB라고 할 수 있다. 예를 들어 E = A = 100이라면, 128KB의 heap 공간이 필요하게 된다. 이러한 heap 공간을 가장 많이 요하는 Primitive - 모듈로는, DOM 트리를 생성하는 XMLPT와 문서에서 사용되는 엘리먼트, 속성 및 개체(entity) 등을 정의하는 DTD 관련 XMLDT가 있다. 또한, XML 문서의 브라우징을 위해, XSL 혹은 XSLT 방식으로 작성된 스타일 문서의 파싱 및 템플레이트(template) 트리 생성, 그리고 HTML 문서로의 변환을 담당하는 Primitive-모듈인 XSLTP가 있다. 그러므로, 대부분의 임베디드 XML 처리기는 이처럼 많은 heap 공간을

<표 4> EmXJ에서 정형(wellformed) 및 적합(valid) XML 처리기의 구성

	MM (1)	MM (2)	MM (3)	MM (4)	MM (5)	MM (6)	MM (7)	MM (8)	MM (9)	MM (10)	XMLDT	XSLTP	Size (KB)
정형 XML 파서	●												15.8
정형 DOM 처리기										●			56.3
정형 SAX 처리기						●							28.9
적합 XML 파서	●										●		44.3
적합 DOM 처리기										●	●		84.8
적합 SAX 처리기						●					●		57.4

필요로 하는 기능을 제거한 SAX 처리기를 사용하고 있으며, DTD를 기반으로 하는 적합 XML 처리기나 DOM 트리의 생성을 필요로 하는 DOM 처리기를 위한 기능들은 지원하지 않는 것이 보통이다.

구 성 요 소		설 명
String	elementName	element의 이름
int	level	element의 레벨
ElementNode	parentElement	element의 parent element 노드
ElementNode	firstChildElement	element의 first child element 노드
ElementNode	preSiblingElement	element의 previous sibling element 노드
ElementNode	postSiblingElement	element의 next sibling element 노드
AttributeNode	firstAttribute	element의 first attribute 노드
TextNode	text	element의 text child 노드

(a) ElementNode의 구조

```
public class ElementNode {
    String elementName = null ;
    int level = 0 ;
    ElementNode    parentElement = null ;
    ElementNode    firstChildElement = null ;
    ElementNode    preSiblingElement = null ;
    ElementNode    postSiblingElement = null ;
    AttributeNode  firstAttribute = null ;
    TextNode       text = null ;

    public ElementNode(String tagName, int tagLevel)
    {
        elementName = tagName ;
        level = tagLevel ;
    }
}
```

(b) ElementNode Java 객체 - 702 Byte>

(그림 8) DOM 트리를 구성하는 하나의 엘리먼트 노드 구조 및 구현된 Java 객체

4.2 소요 메모리 공간 및 문자 처리 속도 비교

잘 알려진 임베딩용 경량급 SAX 처리기인 MinML과 AElfred, 그리고 EmXJ 상에서 구성 가능한 SAX 처리기에 대한 기능 및 코드 크기를 보여주고 있는 것이 <표 5>이다. MinML의 경우 단지 엘리먼트 및 문자 데이터 처리를 위한 2개의 API만 지원하고 있어, 속성 리스트를 가지는 XML 문서에 대한 처리는 할 수 없다. 또한 응용에서의 오류 처리가 이루어질 수 없으며, 파서에서 처리해 주는 오류 메시지에 의존할 수밖에 없는 매우 단순한 SAX 처리기이나, 문자 처리 속도는 매우 빠른 것으로 알려져 있다[4]. AElfred[5]의 경우 11개의 SAX API를 지원하고 있어, 대부분의 SAX 2.0 사양을 만족한다고 볼 수 있으나, 그 코드 크기는 MinML에 비해서 매우 큰 편이라 할 수 있다. 그러나, MinML이 엘리먼트와 문자 데이터 처리만 가능한데 반해, AElfred는 속성 리스트 처리 및 응용에서의 오류 처리 등이 가능하다는 장점을 가진다. MinML이나, AElfred 공통적으로 기능이 경직되어 있어, 활용 메모리 공간이 충분하여 DOM 처리기능을 요하는 고급 장치에는 사용할 수 없다는 문제점을 가지고 있다. 이러한 문제에 용이하게 대처할 수 있다는 것이 EmXJ의 장점이다.

MinML은 엘리먼트와 문자 데이터 처리의 2가지 API만 지원하는 정형 SAX 처리기이다. 즉, DTD를 전혀 지원하고 있지 않다. AElfred는 11개의 API를 지원하는 적합 SAX 처리기이나, 개체(entity)에 대한 DTD 참조만 가능하며, 엘리먼트 혹은 속성에 대한 참조는 지원하고 있지 않다. 그에 비해 Crimson은 적합 DOM 처리기이며, <표 7>에서 보여준 바와 같이 SAX 2.0 및 DOM 레벨 2 Core를 지원하고 있다. 이들 각각에 대해 문서처리 속도를 비교해 놓은 것이 <표 6>이다. EmXJ의 경우 MM(6)의 정형 SAX 처리기를 구성하여 결과를 얻었으며, Crimson의 경우에는 SAX 2.0 파서를 사용하였다. 결과를 얻기 위해 사용된 XML 문서는 엘리먼트와 문자 데이터 그리고 속성 리스트를 포함하고 있으며, 30회 반복하여 평균을 구한 것이다. <표 6>에서 보아 알 수 있듯이 문자 처리속도는 약 22.3kcps(char/sec)로 MinML이 가장 빠르게 나타나고 있음을 알 수 있다. 그리고 EmXJ의 MM(6)와 Crimson이 비슷한 결과를 보여주고 있음을 알 수 있다. 이는 MinML의 경우 엘리먼트와 문자데

〈표 5〉 SAX 2.0 기반 경량급 XML 처리기의 코드 공간 비교

구 분	MinML	AElfred	EmXJ
사용 언어	Java	Java	Java
파서 모듈 크기	18.1 KB	36.0 KB	15.8 KB
SAX 모듈 크기	0.7 KB	43.4 KB	가변(7.8~13.1) KB
전체 크기	18.8 KB	79.4 KB	가변(23.6~28.9) KB
비 고	• 2개의 SAX API 지원 (엘리먼트와 문자데이터)	• 11개의 SAX API 지원 (엘리먼트, 문자데이터, 속성리스트, 오류처리 등)	• 6개의 SAX API 지원 • PM 추가에 의한 지원 API 확장 가능 (MM(3)-MM(6))

〈표 6〉 MinML 기능을 기반으로 한 각 SAX 처리기에서 문자 처리 속도 비교

	EmXJ(MM(6))	Crimson	AElfred	MinML
정형 SAX 처리기	1783ch / 130ms	1783ch / 110ms	1783ch / 491ms	1783ch / 78ms
	약 13700 cps	약 16600 cps	약 3600 cps	약 22300 cps
DTD 분석기	418ch / 50ms	418ch / 60ms	418ch / 221ms	지원하지 않음
	약 8400 cps	약 7000 cps	약 1900 cps	

이터에 대한 이벤트 구동만 가능하며, 속성 리스트는 인식을 못하며 건너뛰기 때문이다. 그러나, EmXJ, Crimson, 및 AElfred 등의 SAX 처리기는 엘리먼트와 문자 데이터 그리고 속성 리스트 모두를 인식하여 이벤트를 구동시키므로 처리 속도가 MinML에 대해 떨어지게 된다. 즉, 지원하는 기능이 많을수록 문자 처리 속도는 떨어지는 현상을 보여주고 있다. 그리고, DTD를 지원하는 각 처리기에 대해 문자 처리 속도를 비교를 보면, 정형 SAX 처리기와 유사한 처리 속도를 보이고 있음을 알 수 있다. 이는 DTD를 기반으로 하는 적합 SAX 처리기의 문자 처리 속도 또한 유사한 관계를 보여줄 것이다.

<표 7>은 Apache사에서 개발된 3가지 유형의 고급 XML 처리기를 보여주고 있으며, 각 부분별 코드 크기는 알려져 있지 않다. 이들의 목적은 다양한 기능을 지원하는 XML 처리기들의 개발을 통해, 다양한 규모의 웹서버 및 브라우저에서 사용할 수 있도록 함을 목적으로 하고 있다[6]. 이들은 모두 SAX API와 DOM API를 부분적으로 혹은 전체로 지원하고 있으며, 그 외 필요하다고 생각되는 기능들이 서로 다르게 추가되어 있다. 이러한 방식의 개발은 많은 인력과 비용을 필요로 하고 있다. 이러한 문제점을 해결해 줄 수 있는 방법이 EmXJ가 지향하는 방향이다. 현재의 EmXJ 상에서 구성 가능한 XML 처리기 수준은 Crimson 정도이나, 다양한 기능의 Primitive-모듈 및 Macro-모듈들을 개발하여, 응용에서 요구하는 기능 및 활용 가능한 자원 등을 고려하여 XML 처리기를 구성할 수 있기 때문이다. 각 장치마다 서로 다른 XML 응용을 요구하고 있고, 그로 인해, 서로 다른 기능을 필요로 하는 경우, 계속적인 기능 변화에 대처하기 위해, XML 처리기는 경직화 될 수 없으며, 확장성 및 유연성을 가지는 XML 처리기가 필요하다. EmXJ는 이러한 환경을 위한 XML 처리기 구성을 위한 프레임워크이다.

5. XML 처리기 구성 및 응용 예제

본 장에서는 EmXJ로부터 구성된 XML 처리기를 사용한 간단한 응용 예를 보여준다.

5.1 MM(3)와 MM(4) 기반의 단순 SAX 처리기

구성 테이블로부터 구성된 MM(3)와 MM(4)를 기반으로 하는 간단한 응용 프로그램을 작성하여, 그 기능의 차이를 보여주고 있는 것이 (그림 9)와 (그림 10)이다. MM(3)와 MM(4)의 차이를 보여주기 위해, (그림 9)의 3번째 줄에서 문장 마침을 의미하는 마침 태그인 </name>이 생략된 오류를 포함하는 XML 문서를 사용하였다. (그림 9)는 엘리먼트와 문자 데이터에 대해서만 이벤트를 발생시키고 해당 이벤트에 대한 API가 응용 프로그램에서 콜백 함수로 구동되도록 지원하는 MM(3) 기반의 SAX 처리기의 동작을 보여주고 있다.

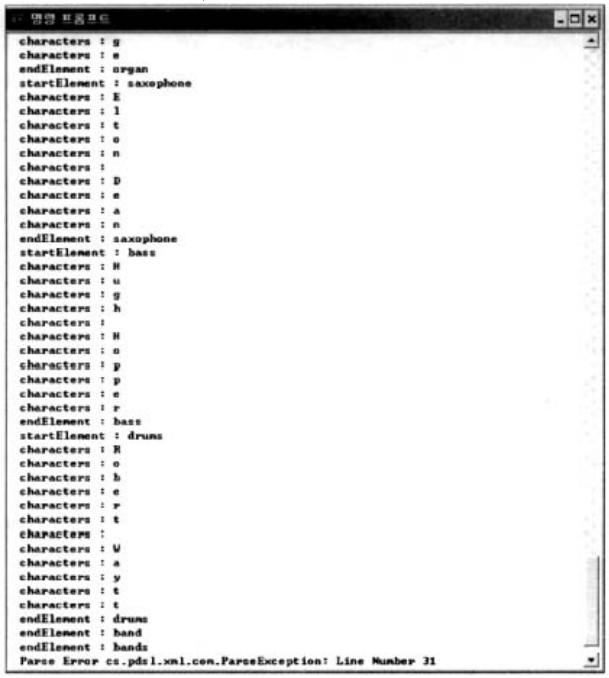
MM(3)의 경우, <표 3>에서 기술된 바와 같이, 오류 발생에 대한 이벤트 구동 모듈이 없으므로, 응용에서 해당 API를 사용할 수 없게 된다. 그러므로, 오류가 발생할 경우, 전적으로 파서의 오류 처리 과정에 의존할 수밖에 없다. 즉, 문서 상 오류가 발생할 경우 응용 프로그램에서는 대처할 수가 없는 것이다.

(그림 10)은 (그림 9)의 MM(3) 기반의 SAX 처리기의 기능에 오류 처리 이벤트 구동 모듈을 추가한 MM(4) SAX 처리기 기반의 MM(3)의 경우와 동일한 응용을 보여주고 있다. <표 3>에서 보여준 바와 같이, MM(4)의 경우는 오류 발생에 대한 이벤트 구동 모듈을 포함하고 있으므로 오류가 발생할 경우, 파서에서의 오류 처리 과정이 부족할 경우, 오류 처리 API를 사용하여, 응용 프로그램에서 적절한 대응을 할 수 있다. 본 예제에서는 응용 프로그램에서의 오류 처리를 간단한 메시지를 출력하도록 하였다.

〈표 7〉 Apache사에서 개발된 3가지 유형의 XML 처리기 비교 분석

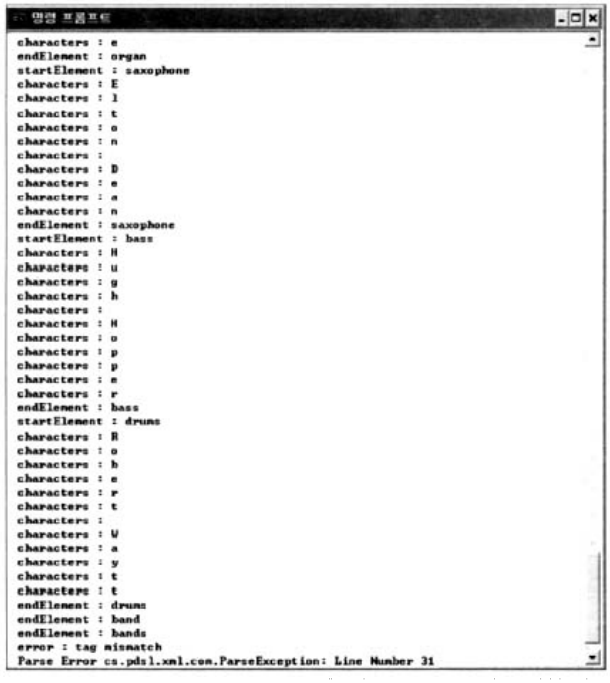
구 분	Xerces	Crimson	Xalan
사용 언어	Java	Java	Java
처리기 크기	1.7MB(jar 파일)	201KB(jar 파일)	3.9MB(jar 파일)
비 고	• XML 1.0 및 SAX 2.0 • DOM level 2 Core, Events, Traversal/Range • Namespace in XML • XML Schema 1.0	• XML 1.0 및 SAX 2.0 • DOM level 2 Core	• XML 1.0 및 SAX 2.0 • DOM level 2 5개 모듈 • XSLT 지원 • XPath 지원

```
<bands>
  <band type = "progressive">
    <name> King Crimson
    <guiter> Robert Fripp </guiter>
    <saxophone> Mel Collins </saxophone>
    <bass> Boz </bass>
    <drums> Ian Wallace </drums>
  </band>
  <band type = "punk">
    <name> X-Ray Spex </name>
    <vocals> Poly Styrene </vocals>
    <saxophone> Laura Logic </saxophone>
    <guiter> Someone else </guiter>
  </band>
  <band type = "classical">
    <name> Hilliard Ensemble </name>
    <saxophone> Jan Garbarek </saxophone>
  </band>
  <band type = "progressive">
    <name> Soft Machine </name>
    <organ> Mike Ratledge </organ>
    <saxophone> Elton Dean </saxophone>
    <bass> Hugh Hopper </bass>
    <drums> Robert Waytt </drums>
  </band>
</bands>
```



(그림 9) MM(3) SAX 처리기의 간단한 응용 예

```
<bands>
  <band type = "progressive">
    <name> King Crimson
    <guiter> Robert Fripp </guiter>
    <saxophone> Mel Collins </saxophone>
    <bass> Boz </bass>
    <drums> Ian Wallace </drums>
  </band>
  <band type = "punk">
    <name> X-Ray Spex </name>
    <vocals> Poly Styrene </vocals>
    <saxophone> Laura Logic </saxophone>
    <guiter> Someone else </guiter>
  </band>
  <band type = "classical">
    <name> Hilliard Ensemble </name>
    <saxophone> Jan Garbarek </saxophone>
  </band>
  <band type = "progressive">
    <name> Soft Machine </name>
    <organ> Mike Ratledge </organ>
    <saxophone> Elton Dean </saxophone>
    <bass> Hugh Hopper </bass>
    <drums> Robert Waytt </drums>
  </band>
</bands>
```



(그림 10) MM(4) SAX 처리기의 응용 예

5.2 MM(9) 기반의 DOM 처리기

(그림 11)은 엘리먼트, 문자 데이터, 그리고 속성 리스트를 포함하는 오류가 없는 XML 문서와 그에 대해 구문 검사 및 DOM 트리를 생성하는 MM(9) 기반의 DOM 처리기를 사용하는 간단한 응용 예이다. DOM 트리의 생성을 보여주기 위해 오류가 없는 문서를 사용하였다. 문서에서 사용된 엘리먼트에 대한 DOM 트리의 형태 생성 형태를 (그림 11)에서 보여주고 있으며, 속성과 텍스트에 대해서는 해당 엘리먼트에 연결되어 있는 형태로 표현하고 있다.

6. 결론 및 향후 연구

유무선 인터넷의 발달로, 다양한 형태의 자원제약을 가지는 개인용 혹은 상업용 임베디드 시스템들이 속속 등장하고 있으

```
<bands>
  <band type = "progressive">
    <name> King Crimson </name>
    <guiter> Robert Fripp </guiter>
    <saxophone> Mel Collins </saxophone>
    <bass> Boz </bass>
    <drums> Ian Wallace </drums>
  </band>
  <band type = "punk">
    <name> X-Ray Spex </name>
    <vocals> Poly Styrene </vocals>
    <saxophone> Laura Logic </saxophone>
    <guiter> Someone else </guiter>
  </band>
  <band type = "classical">
    <name> Hilliard Ensemble </name>
    <saxophone> Jan Garbarek </saxophone>
  </band>
  <band type = "progressive">
    <name> Soft Machine </name>
    <organ> Mike Ratledge </organ>
    <saxophone> Elton Dean </saxophone>
    <bass> Hugh Hopper </bass>
    <drums> Robert Waytt </drums>
  </band>
</bands>
```

(그림 11) DOM 처리기(MM(9)) 응용 예

며, 이들로 임베딩되는 소프트웨어는 경직성보다도, 자원 제약 특성이 서로 다른 장치들로 유연한 임베딩을 위한 특성을 요구하고 있다. XML 처리기는 인터넷 상의 XML 자료 교환을 위하여 각종 인터넷 단말에 필수적으로 임베딩 되어야 할 소프트웨어 중 하나이며, 본 논문에서는, 가변적 기능과 자원 제약 특성을 가진 다양한 형태의 단말 장치로의 유연한 임베딩이 가능한 XML 처리기의 구성을 위한 프레임워크인 EmXJ가 설계, 구현되었으며. 기존의 다른 XML 처리기와 소요 메모리 공간, 기능 및 처리 속도 등이 비교 분석되었다.

EmXJ를 기반으로, 기능 및 소요 메모리 공간이 각가 다른 단순 기능을 가지는 경량급 XML 처리기(예를 들면, MM(3))로부터 SAX 및 DOM까지의 기능을 가지는 중량급 XML 처리기(예를 들면, $MM(6) \cup MM(10)$)까지 다양한 XML 처리기를 구성할 수 있음을 보여 주었다. 이는 기능의 확장 혹은 변화가 빠른 장치에 임베딩될 XML 처리기를 효율적으로 구축할 수 있으며, 또한 계층구조를 가지고 있으므로, 계속되는 XML 사양의 확장 및 변경에 효율적으로 대처할 수 있는 기반을 제공하고 있다. 그러므로, XML 처리기를 기반으로 하는 다양한 응용에 효율적으로 사용될 수 있다는 장점을 가진다. 현재, EmXJ는 SAX 2.0 및 DOM 레벨 2에서 제정된 모든 기능을 지원하지는 못하고 있으므로, 향후, 완전한 기능을 위하여 계속적인 Primitive-모듈과 Macro-모듈의 추가를 통해 PM 계층과 MM 계층을 확장 시켜야 할 것이다.

참 고 문 헌

[1] W3C, http://www.w3c.org/TR/REC-xml/.

[2] R. Tobin, RXP, http://www.cogsci.ed.ac.uk/~richard.

[3] J. Clark, expat, http://www.jclark.com./xml/expat.htm.

[4] MinML, http://www.wilson.co.uk/.

[5] AElfred, http://www.microstar.com/XML/.

[6] Crimson, Xerces, Xalsn, http://xml.apache.org/.

[7] XML4J, http://alphaworks.ibm.com/text/xml4j.

[8] E. A Lee, "What's Ahead for Embedded Software?," IEEE Computer, Vol.33, No.9, Sept., 2000.

[9] S. Ortiz Jr., "Embedded OSs Gain the Inside Track," IEEE Computer, Vol.34, No.11, Nov., 2001.

[10] A. D. Pimentel et al, "Exploring Embedded-Systems Architectures with Artemis," IEEE Computer, Vol.34, No.11, Nov., 2001.

[11] D. Mulchandani, "Java for Embedded Systems," IEEE Internet Computing, Vol.2, No.3, May/June, 1998.

[12] A. Abrardo and A. L. Casini, "Embedded Java in a Web-Based Teleradiology System," IEEE Internet Computing, Vol.2, No.3, May/June, 1998.

[13] N. Witchey, "Designing an Embedded Web Server," IEEE Internet Computing, Online, 1998.

[14] http://www.virata.com/products/emserver1.htm, "Embedded Web Server," 2000.

[15] http://www.virata.com/products/emserver2.htm, "Embedded Web Server Specifications," 2000.

[16] 정원호 외, "구성적 임베딩을 위한 모듈 기반의 XML 처리기의 설계", 한국정보과학회 춘계학술대회논문집, 2002.

[17] W3C, http://www.w3c.org/TR/xsl/.

[18] W3C, http://www.w3c.org/TR/xslt/.

[19] B. R. Montague, "JN : OS for an Embedded Java Network Computer," IEEE Micros, May/June, 1998.

[20] B. Day, "Developing Wireless Applications Using the Java2 Platform, Micro Edition," a white paper, http://java.sun.com/j2me.

정 원 호

e-mail : whchung@center.duksung.ac.kr
1979년 서울대학교 전자공학과(학사)
1981년 한국과학기술원 전기 및 전자공학과 (석사)
1989년 한국과학기술원 전기 및 전자공학과 (박사)
1979년~1982년 대한전선(주)
1983년~1984년 대우통신(주)
1993년 IBM T. J. Watson 연구소 방문 연구원
1989년~현재 덕성여자대학교 컴퓨터과학부 교수
관심분야 : 분산 및 병렬처리, 모바일컴퓨팅, 임베디드 시스템 등

강 미 연

e-mail : mykang@icantek.com
1999년 덕성여대 전산학과(학사)
2001년 덕성여대 전산 및 정보통신학과 (석사)
1999년~2001년 덕성여대 전산학과 조교
2001년~현재 ICANTEK(주) 기술연구소
관심분야 : 이동 에이전트 시스템, 임베디드 시스템 등