추상구문트리를 이용한 구문지향 XML 문서 편집기

A Syntax-Directed XML Document Editor using Abstract Syntax Tree

김 영 철* 유 두 규**
Young-Chul Kim Do Kyu You

요 약

기존의 XML 문서 편집기는 일반 텍스트 위주의 편집을 하고 내부적으로 구문적 검사를 하지 않는다. 따라서 작성된 XML 문서가 잘 설계(well-formed) 되었는지 유효(valid) 문서인지를 검사하지 못한다. 본 논문에서는 XML 문서를 편집하는데 있어서 구문에 맞도록 설계할 수 있는 구문지향 편집기를 설계하고 구현한다. 또한 트리 기반의 편집기로 구현되어 있기 때문에 향후에 XML 문서 확장이 용이하며, 다른 시스템과는 달리 실시간으로 유효성을 검증할 수 있도록 설계되었다. 본 시스템은 향후에 XML 관련 어플리케이션 개발에 많은 영향을 줄 것으로 기대된다.

Abstract

The current text based XML document systems are editing text and don't perform syntax check. As a result, the validity of an edited XML document can't be decided it is well-formed or valid until it is parsed. This paper describes a design and implementation of the syntax-directed editing system for XML documents. Because this is tree-based system, it is easy to extend XML document. And this system is designed to validate XML documents in real-time. It is expected that this paper contributes XML related application developments.

☞ Keyword : AST, XML, Syntax-Directed Editor, DTD

1. 서 론

1.1 연구 배경 및 목적

XML은 W3C에서 제정한 표준이며 SGML(Standard Generalized Markup Language)의 부분집합이라고 할 수 있으며 SGML보다 간결화된 문서를 표현하는 표준이다. 즉, XML은 SGML의 방대한 스펙 중 복잡하고 사용하지 않는 것들을 없애고, 고정된 DTD를 사용하는 스타일 중심의 HTML의 단점을 보완하고자 개발되었다[1].

XML은 HTML과 유사한 방법으로 문서를 표현하는데, HTML에서 오직 지정된 태그(tag)만을 사용할 수 있지만, XML에서는 엘리먼트(element)를 사용자가 자유롭게 지정할 수 있다. 즉, 문서의 논리적 구조를 표현하기 위해 작성자가 문서에 의미를 부여할 수 있다는 것이다. 따라서 XML은 그 자체만으로는 HTML과 같이 웹상에 표현이 되지 않으며 XML을 지원하는 브라우저로 볼 경우, 그 논리 구조만이 브라우저상에 나타난다. 이것은 HTML과 달리 스타일 언어를 따로 분리해두었기 때문이다. 따라서 XML을 HTML과 같은 형식으로 표현하기 위해서는 XSL이란 스타일 언어가 필요하다. 또한 XML은 HTML에서 제공하는 방식의 링크의 개념을 벗어나 단순링크(simple link), 확장링크(extended link) 등과 같은 다양한 링크를 제공한다[2,3].

구문지향 편집기는 프로그램 중심적인 편집, 번역, 실행, 디버깅 기능이 포함되어 프로그램 편집

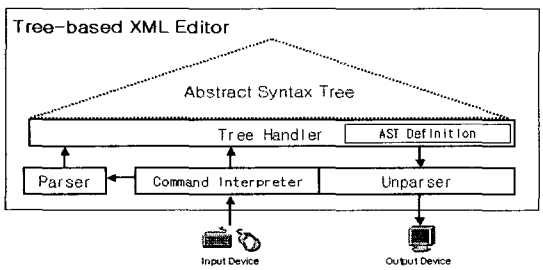
* 정 회 원 : 숭실대 정보미디어기술연구소
yckim@ss.ssu.ac.kr(제 1저자)
** 정 회 원 : 세명 컴퓨터 고등학교
bima@dreamwiz.com(공동저자)
[2004/03/05 투고 - 2004/03/26 심사 - 2005/02/01 심사 완료]

과 컴파일러의 구문 분석 및 점진적 의미 분석 기능을 동시에 수행할 수 있는 대화식 편집기이다 [4-6]. 구문지향 편집기를 이용한 프로그램 편집은, 첫째 사용자가 프로그래밍 언어의 어려운 문법 규칙에 구속되지 않고, 둘째 프로그램이 하는 일과 방법, 결과, 이유 등의 직관적 이해로 프로그램 논리의 설계 및 정확성 확인에 집중할 수 있어 소프트웨어 생산성을 향상시키고, 셋째 새로운 언어에 대한 접근을 쉽게 하는 장점이 있다.

구문지향 편집기는 프로그램이 항상 구문적으로 잘 설계된(well-formed)인지 확인하기 위해 언어의 구문 지식을 사용한다. 사용 언어의 지식은 첫째 구문 지식을 적용하여 컴파일시간보다 프로그램 입력 시간에 오류를 검사한다. 둘째, 언어의 의미 지식은 편집하는 시간에 점진적으로 의미 분석 및 오류를 진단하고 목적 코드를 생성함으로써 프로그램 전체를 재 컴파일 하는 등의 지연시간이 없고, 인터프리터에 의해 즉각적인 실행과 디버깅을 할 수 있는 기능 등을 제공한다. 셋째, 화면에 출력되는 정적의미 분석 결과는 프로그램을 개발, 수정하는 동안 프로그래머에게 즉각적인 피드백을 제공한다. 넷째, 이러한 일련의 컴파일 과정은 프로그램의 수정에 따라 영향을 받는 최소한의 범위 내에서만 효율적으로 이루어 질 수 있도록 하였다.

본 논문에서 구현한 편집기는 그림 1과 같은 시스템 구조를 갖는다. 다음은 편집기 각각의 구성요소에 대한 설명이다.

ASTD(Abstract Syntax Tree Definition)는 DTD를 내부 편집기나 역파서에서 사용할 수 있도록 만들어놓은 문법 트리이다. 편집기에서 만들어지는 모든 문서는 ASTD에 의해서 문법이 검증되어 올바른 문서만 만들어진다. 또한 추상구문트리(Abstract Syntax Tree)는 ASTD를 통한 유효성 검증을 거쳐서 만든 내부 자료구조이다. 파서(Parser)는 문서 유효성 검증을 한다. 파서는 문서의 토큰을 인식해 내는 어휘 분석기와 일련의 토큰의 조합이 적합한 구문 구조를 갖는가를 검사하는 구문 분석기로 구성된다. 본 시스템에서의 역파서(Unparser)는 AST를 화면상에 변환하여 보여주는 역할을 하며, AST의 각 노드를 그래픽 트리 구조로 보여준다. 또한 역파서는 AST를 그래픽 트리 구조로 역파싱하는 기능 외에 일반 텍스트 문서로 저장도 한다. 트리 조작기(Tree Handler)는 ASTD를 참조해서 실제로 문법이 올바른 AST 노드를 추가/삭제/갱신에 대한 제어를 한다. 명령어 처리기(Command Interpreter)는 사용자가 편집 환경에서 입력한 명령어를 해석하는 부분으로 입력된 명령어가 수행 가능한 것인지를 검사하고 수행 가능한 명령어인 경우에는 해당 기능을 수행한다. 이때 각각의 명령은 트리 조작기를 통해서 실제로 내부 트리 연산이 발생하게 된다.



〈그림 1〉 전체 시스템 구조도

본 논문은 다음과 같은 형태로 구성되어 있다. 본 논문의 2장에서는 시스템을 설계 및 구현하기 위한 이론적 배경을 설명하고, 3장에서는 관련 편집기 구현 사례를 고찰하고, 4장에서는 전체 시스템을 설계하고 이를 구성하는 각 부분의 기능을 명세하며, 이들 간의 인터페이스를 규정하고 이를 구현한다. 마지막으로 5장에서는 논문의 결론을 맺는다.

2. 관련 연구

XML을 위한 편집 시스템은 현재 여러 가지 언어로 제작되고 있는데, DTD를 가지고 유효성 검사(validating)을 하는 편집기와 DTD없이 잘 설계(well-formed) 되었는지 만을 검사하는 편집기로 나뉘어 진다. 또한 현재의 편집기들은 텍스트 편집 환경보다는 트리와 같은 그래픽 사용자 인터페이스를

사용해서 보다 직관적으로 편집이 가능하다. 그러나 이들은 XML 문서처리를 완벽하게 처리하지 못하거나, XML의 변형된 형태의 문서를 처리하거나, SGML에 관련된 소프트웨어를 수정한 형태를 가짐으로서 순수한 XML 문서를 처리하는데 적합하지 못하며, XML 어플리케이션 사용자를 위한 소프트웨어로 부족한 점이 많다. 기존의 XML 문서 편집기로는 Techno2000에서 개발한 CLIP! XML Editor 1.5[7]이 있으며, Vervet Logic에서 개발한 XML Pro v2.0[8], Cuesoft의 EXml[9]이 있다.

CLIP! XML Editor는 현재 버전 1.5까지 개발되었으며 트리 기반, 텍스트 기반의 편집을 지원하며 새문서 작성 마법사를 통해서 처음부터 마지막 단계까지 단계별로 가이드 해주는 마법사를 이용한 문서 저작을 지원한다. 또한 DTD 트리 보기 기능, 에러 출력 및 수정 기능을 제공하며 잘 설계된 XML 문서에서 DTD를 추출, 생성해 준다.

XML Pro v2.0은 순수 자바 애플리케이션으로 쉽고 직관적인 그래픽 사용자 인터페이스, 문서 구조를 유지하는 문서 트리 편집 뷰, DTD를 통한 XML 유효화 검증을 지원한다. 또한 편리한 엘리먼트 생성 및 관리를 위한 엘리먼트 마법사(element wizard), 속성 생성 및 관리를 위한 속성 마법사(attribute wizard)도 제공된다. 기본적으로 엔티티(entity), CDATA, 주석을 포함한 W3C의 XML 1.0 스펙을 지원한다.

Cuesoft의 EXml은 well-formed XML 문서 편집기로서 트리나 소스 뷰를 지원한다. 트리 뷰를 통해서 엘리먼트 편집이 가능하다. 다른 편집기와 마찬가지로 well-formedness를 체크하는 것 이외에 특이한 사항은 XSL 이름공간(namespace)지원과 PCDATA, 주석, 속성값을 엘리먼트에 매핑한 테이블 출력, 소스뷰 상에서의 직접접적인 텍스트 편집이 가능하다는 것이다.

본 논문에서는 제시한 시스템은 앞서 기술한 시스템에 비해 크게 2가지 장점을 가지고 있다. 첫 번째는 XML 어플리케이션 DTD를 가지고 문서 인스턴스(document instance)를 파싱함으로써 구문분석(syntax analysis)과 의미 분석(semantic analysis)이 이루어져, 문서가 잘 설계(well- formed) 되었는지와 유효(valid)한 문서인지를 검사한다. 두 번째는 XML 어플리케이션 문서를 시각화하기 위해서 XML 추상구문트리 정의의 역파싱 규칙(unparsing scheme)을 이용하는데, 이를 위해서 XML 문서를 처리해서 생성된 문서 인스턴스 트리 정보와 역파싱 규칙을 매핑시켜 화면에 출력하도록 하는 것이다. 이는 문서 편집 시에 문서 시각화와 시각 프로그래밍을 동시에 제공하여 각각이 지니는 단점을 상호보완하고, 개발자에게는 일관된 시각기호를 제공하므로써 개발 대상에 대한 인지도를 높이고자 한다. 이러한 시스템에서 갖추어야할 조건으로는 먼저 사용하는 시각기호가 문서 시각화와 시각 프로그래밍 모두에 적절해야 한다는 점과 시스템에 생길 확장 등을 고려하여 시스템이 유연하게 설계되어야 한다는 점을 들 수 있다.

3. 본 론

3.1 추상 구문 트리(Abstract Syntax Tree)

AST를 구성하는 노드의 구조는 표 1과 같다. 이 노드는 문서 내에서 사용되는 태그 이름과 연관된 속성정보와 AST에서 다른 노드들과의 연결정보들로 구성된다. 문자열 데이터는 작성자가 직접 작성한 문서 내용을 갖고 있는 부분으로, 이것은 PCDATA, CDATA나 주석과 같은 서브 엘리먼트로 이루어진 엘리먼트인 경우에만 유효한 정보이다. 실제 이 부분은 문자열만을 저장한다.

〈표 1〉 AST 노드 자료구조

생성 번호				
문자열 데이터				
속성 리스트				
부모 노드연결	오른쪽 노드 연결	왼쪽 노드 연결	다음 노드 연결	이전 노드 연결

AST 노드의 연결정보는 5가지로 구분한다. 먼저 물리적인 노드의 연결 정보로 부모, 오른쪽, 왼쪽 노드 연결 정보가 있다. 문서의 AST는 편집기에서 항상 이용되는 문서 정보이므로, 편집기에서 작성자가 선택할 수 있는 문서 태그 위치에 제한을 두지 않게 하기 위하여 이전 노드와 다음 노드 연결 정보를 추가하였다. 이전 노드 연결과 다음 노드 연결의 기본적인 트리 순회방식은 프리오더(pre-order) 방식을 이용한다.

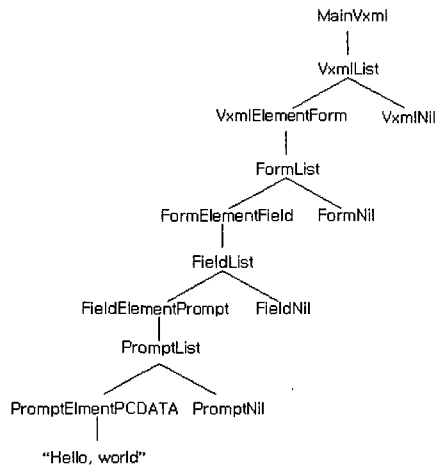
AST는 XML 문서를 파싱하여 편집기에서 편집 가능하도록 설계한 자료구조이다. 사용자가 편집기에서 행하는 편집 연산에 대한 처리는 모두 AST 인스턴스에 반영된다. 예를 들면, 다음 표 2와 같은 XML 문서가 있다면, 그림 2와 같은 AST 인스턴스를 생성하게 된다. AST 인스턴스는 XML 문서를 파싱하고 표 3과 같이 DTD를 변환한 ASTD를 이용하여 각각의 엘리먼트에 노드와 중간 임시 노드를 생성하게 된다.

3.2 추상 구문 트리 정의(ASTD: Abstract Syntax Tree Definition)

편집기의 핵심은 DTD를 문법 규칙의 집합인 추상 문법(abstract syntax)으로 정의하는 것이다. 편집하는 개체(object)는 문법에 따라서 유도 트리

〈표 2〉 XML 문서

```
<?xml version="1.0">
<!DOCTYPE vxml SYSTEM "vxml.dtd">
<vxml version="1.0">
 <form>
 <field>
 <prompt>
 Hello, world
 </prompt>
 </field>
 </form>
</vxml>
```



〈그림 2〉 XML문서의 AST 인스턴스

(deprivation tree), 즉 AST로 표현된다. 편집기 사용자 인터페이스의 조작에 따른 텍스트나 문서 구조의 변화는 주어진 구문 트리가 변화함을 의미한다. 본 논문에서는 XML 어플리케이션 중에서 VoiceXML의 DTD를 ASTD로 변환하였다. 추상 문법은 다음과 같은 형식의 프로덕션(production)의 집합으로 이루어진다[6].

$$x_0 : op(x_1, x_2 \cdots x_k);$$

여기서 op는 연산자명(operator name)이고 각 x_i는 문법의 비단말명(nonterminal name)이다. 프로덕션 인스턴스를 구별하기 위한 목적인 연산자를 제외하고 문법 규칙은 다음과 같은 문맥 자유 프로덕션(context-free production)과 같다.

$$x_0 \rightarrow x_1 x_2 \cdots x_k$$

다음의 표 3은 XML 어플리케이션의 하나인 VoiceXML DTD의 일부를 ASTD로 변환한 것이다.

3.3 XML 어플리케이션 DI 파서

XML 문서는 실제로 작성되는 문서 내용으로서

〈표 3〉 DTD의 ASTD변환

```
<!ELEMENT vxml
   (catch | help | noinput |nomatch |
    error | form | link |menu | meta |
    property | script | var)+>

<!ELEMENT form
   (grammar | initial | subdialog |
    field | filled | record | dtmf |
    block | var | catch | help |
    noinput | nomatch | error | object |
    link | transfer | property |
    comment)* >
```

〈VoiceXML DTD의 일부〉

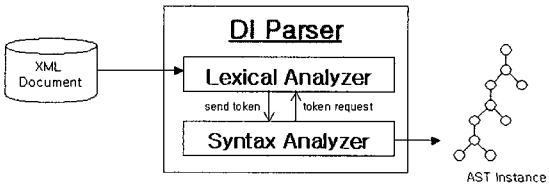
```
main : MainVxml(vxml)
   ;
vxml : VxmlNil( )
   | VxmlList (vxmlElement vxml)
   ;
vxmlElement: VxmlElelementNil ()
   | VxmlElementCatch (catcch)
   | VxmlElementHelp (help)
   | VxmlElementNoinput (noinput)
   | VxmlElementNomatch (nomatch)
   | VxmlElementError (error)
   | VxmlElementForm (form)
   | VxmlElementMenu (menu)
   | VxmlElementLink (link)
   | VxmlElementMeta (meta)
   | VxmlElementProperty (property)
   | VxmlElementVar (var)
   | VxmlElementScript (script)
   | VxmlElementComment (comment)
   ;
```

〈변환된 ASTD〉

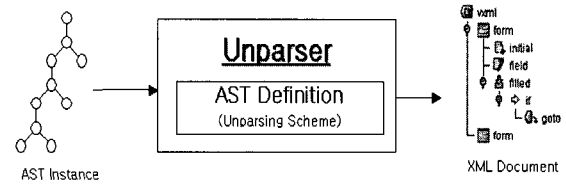
DI (Document Instance)라고 한다. 특히 이 DI는 XML DTD에 따라 종속적으로 작성되는 문서이므로 사용자는 DTD를 숙지해야만 하는 번거로움이 있다. 따라서 특정 DTD를 적용한 XML 어플리케이션 편집기를 사용하면 사용자의 부담이 줄어준다. 본 논문에서 처리하는 트리 형태의 XML 문서 구조를 AST 인스턴스(AST instance)라고 한다. DI 파서는 XML 어플리케이션 문서를 파싱하여 AST 인스턴스를 생성한다.

파서의 구성은 그림 3과 같이 크게 2가지 부분으로서 어휘 분석을 담당하는 어휘 분석기와 구문분석을 담당하는 구문분석기로 구성된다.

어휘 분석기는 DI에서 일련의 문자를 받아서 토큰으로 인식하여 해당 토큰을 구문 분석기에 전달한다. 토큰은 각 엘리먼트의 태그와 속성 및 주석으로 구성되어 있다. 구문 분석기는 어휘 분석기로부터 토큰을 받아서 주어진 토큰의 조합이 문법에 올바른 것인지를 검증한다. 이 때 하향식 파싱(top-down parsing)의 대표적인 방식이며 다중 엔트리(multiple entry)를 지원하는 RDP(recursive descent parsing) 기법을 사용한다. RDP는 결정적 파서(deterministic parser)로서 비결정적 파서(non-deterministic parser) 보다 빠르며, 다중 엔트리를 통해서 문서 일부에 대한 유효성 검증이 가능하다.



〈그림 3〉 문서 인스턴스 파서



〈그림 4〉 역파서

3.4 역파서(Unparser)

역파서는 그림 4와 같이 AST 인스턴스를 입력으로 받아들이고 원래 문서 형태를 ASTD의 역파싱 규칙(unparsing rule)을 참조하여 출력장치(화면 혹은 디스크)에 적당한 형태를 갖추어 보기 좋게 출력한다.

3.5 트리 조작기

트리 조작기는 AST 인스턴스에 대한 직접적인 제어를 담당한다. 이 때 내부 트리를 방문하여 특정 노드를 생성/삭제함은 물론 ASTD를 사용하여 특정 노드에서 확장 가능한 엘리먼트를 찾아낼 수도 있다. 트리 조작기는 표 4와 같은 연산을 수행한다.

〈표 4〉 트리 조작기의 연산

분 류		연 산	설 명
노드 단위 연산	노드 액세스	getNext	다음 노드 방문
		getPrev	이전 노드 방문
		getParent	부모 노드 방문
		getLeft	왼쪽 노드 방문
		getRight	오른쪽 노드 방문
	노드 삽입	insertLeft	왼쪽 노드 삽입
		insertRight	오른쪽 노드 삽입
	노드 삭제	deleteNode	노드 삭제
엘리먼트 단위 연산	엘리먼트 액세스	getNextElement	다음 엘리먼트 방문
		getPrevElement	이전 엘리먼트 방문
		getParentElement	부모 엘리먼트 방문
		getChildElement	자식 엘리먼트 방문
		getNextBrotherElement	다음 형제 노드 방문
		getPrevBrotherElement	이전 형제 노드 방문
	엘리먼트 삽입	insertAsChildElement	현재 엘리먼트에서 자식 엘리먼트로 삽입
		insertAfterElement	현재 엘리먼트 아래로 삽입
		insertBeforeElement	현재 엘리먼트 위로 삽입
	엘리먼트 삭제	deleteElement	엘리먼트 삭제.
	엘리먼트 이동	moveUpElement	엘리먼트를 위로 이동
		moveDownElement	엘리먼트를 아래로 이동
		moveAfterElement	해당 노드 아래로 이동 (드래그 앤 드롭 수행시)
	엘리먼트 편집	cutElement	엘리먼트 자르기
		pasteElement	엘리먼트 붙이기
		copyElement	얼리먼트 복사
	속성 액세스	addAttr	엘리먼트 속성 추가
		removeAttr	엘리먼트 속성 삭제
	문자열 액세스	setStr	엘리먼트 문자열 설정

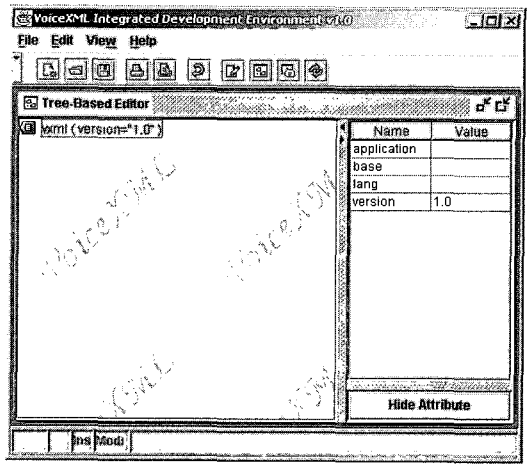
표 4를 보면 트리 조작기의 연산은 노드 단위 연산과 엘리먼트 단위 연산 두 가지로 나누어진다. 전자는 파서에서 직접 모든 노드들을 생성할 때 사용한다. 후자는 기본 노드 단위 연산의 조합으로서 보다 추상화된 연산이며 사용자의 입력을 처리할 때 사용한다.

4. 시스템 구현

본 시스템에서는 XML 문서의 구조적 시각화를 위해서 자바 스윙(JDK 1.2.2에서 구현)에서 제공하는 트리와 테이블 인터페이스를 이용하였다[10].

4.1 초기 화면

그림 5는 트리 기반 XML 어플리케이션 문서 편집기의 초기 새문서로 시작한 그림이다. 윈도우의 좌측은 엘리먼트 트리 윈도우로서(element tree window) XML 문서를 트리 구조로 보여주는 윈도우이다. 문서 작업의 대부분은 엘리먼트 트리 윈도우의 엘리먼트 편집으로 이루어진다. 우측은 속성 윈도우(attribute window)로서 엘리먼트 트리 윈도우에서 엘리먼트를 선택했을 때 해당 엘리먼트의 속성을 편집하는 윈도우이다. 또한 속성 윈도우 밑에는 속성 보이기/숨기기 버튼이 있다. 이

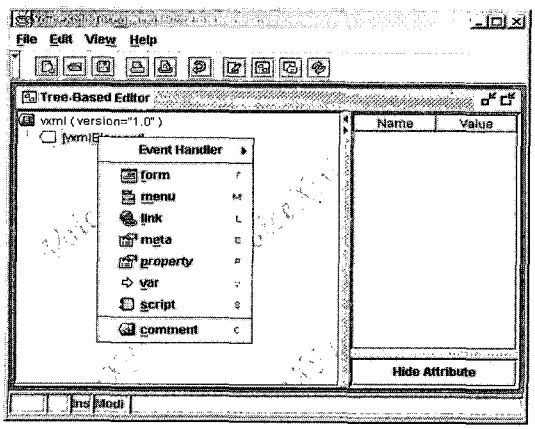


〈그림 5〉 초기 화면

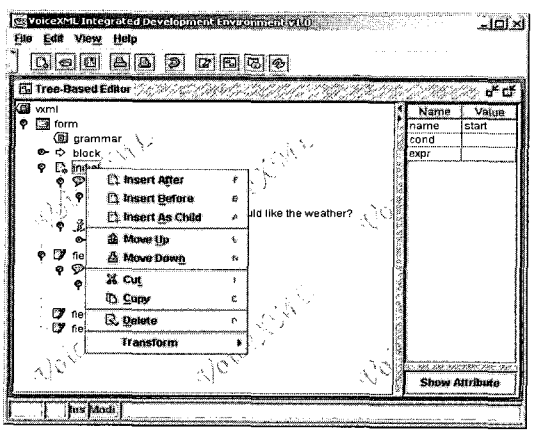
버튼은 좌측의 엘리먼트 트리 윈도우의 각 엘리먼트 노드에 속성 정보를 보이거나 숨기는 기능을 토글 시킨다. 문서의 구조적인 정보는 숨기기 설정하고 세부적인 내용은 보이기로 하는 설정하는 것이 좋다.

4.2 엘리먼트를 위한 인터페이스

다음 그림 6과 7은 엘리먼트 편집을 위한 팝업 메뉴를 보여준다. 엘리먼트 생성을 위해서는 노드에 마우스 왼쪽 버튼을 누르면 그림과 같이 나타난다.



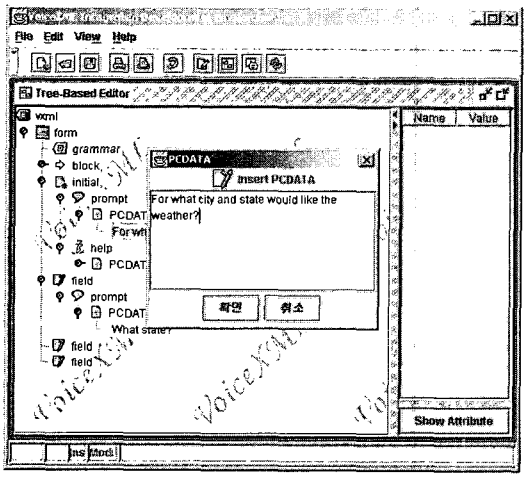
〈그림 6〉 확장 가능한 엘리먼트 팝업 메뉴



〈그림 7〉 엘리먼트 편집 팝업 메뉴

4.3 PCDATA, 주석 편집을 위한 인터페이스

Placeholder를 포함한 모든 엘리먼트는 트리 윈도우에서 편집을 한다. 그러나 텍스트입력을 받는 PCDATA와 주석은 그림 8과 같이 따로 입력 대화상자를 가지고 편집을 한다. 이때 사용자가 입력을 마치고 확인 버튼을 눌렀을 때, 입력한 내용이 유효한지를 검사하기 위해서 내부 다중 엔트리 파서를 사용한다. 입력에 오류가 있을 경우에는 다시 입력을 받는다.



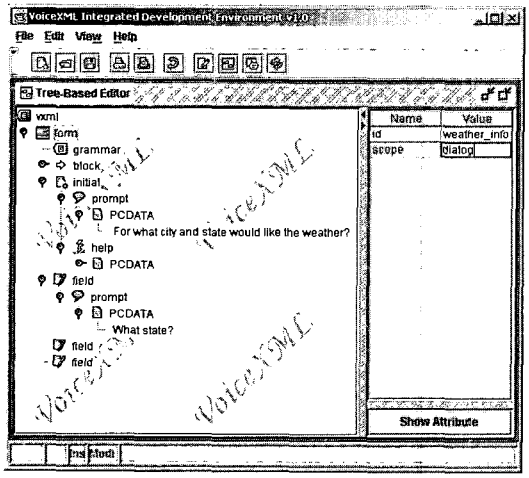
〈그림 8〉 PCDATA 편집

4.4 속성 편집을 위한 인터페이스

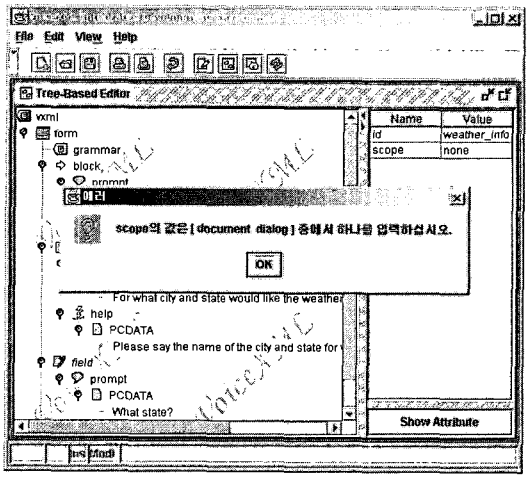
속성 편집을 위한 인터페이스는 다음 그림 9와 같다. 엘리먼트 트리 윈도우의 각 엘리먼트를 클릭하면 이에 오른쪽 속성 윈도우에 해당 엘리먼트에 대한 속성명과 속성 값이 나온다. 이 때 각 값에 대한 편집이 가능하다. 각각의 속성 값에 대한 입력을 마친 후에는 AST 정의의 속성 정보를 이용하여 속성 값 유효성 검증을 한다. 만일 값에 오류가 발생한 경우에는 그림 10과 같이 오류 메시지가 나온다.

5. 결 론

본 논문에서는 추상구문트리(AST)를 이용하여 구문지향 XML 문서 편집기를 구현하였다. 본 시스템에서 개발한 편집기는 다른 시스템과는 달리 XML 어플리케이션 DTD를 가지고 문서 인스턴스(document instance)를 파싱함으로써 구문분석(syntax analysis)과 의미 분석(semantic analysis)이 이루어져, 문서가 잘 설계(well- formed) 되었는지와 유효(valid)한 문서인지를 검사한다. 또한 XML 어플리케이션 문서를 시각화하기 위해서 XML 추상구문트리 정의의 역파싱 규칙(unparsing scheme)



〈그림 9〉 속성 편집



〈그림 10〉 속성 편집 오류

을 이용하였으며, 이를 위해서 XML 문서를 처리해서 생성된 문서 인스턴스 트리 정보와 역파싱 규칙을 매핑시켜 화면에 출력하도록 하였다. 이는 문서 편집 시에 문서 시각화와 시각 프로그래밍을 동시에 제공하여 각각이 지니는 단점을 상호보완하고, 개발자에게는 일관된 시각기호를 제공하므로써 개발 대상에 대한 인지도를 높일 수 있는 효과를 보였다. 또한 XML의 내부 문법이 익숙하지 않더라도 쉽게 사용이 가능하도록 문서 비주얼 프로그래밍 환경을 제공하였다. 또한 DTD를 ASTD와 다중 엔트리를 지원하는 파서를 내장함으로써 사용자들이 문서 편집 시 문법에 맞는 문서를 작성할 수 있도록 함으로써 문서의 오류를 줄였다. 따라서 사용자는 문서 작성 중 항상 유효(valid)한 문서를 작성하게 됨을 알 수 있었다.

향후 연구과제는 DTD의 ASTD 변환을 자동화하고 이를 이용하여 실시간 유효성 검증이 가능한 범용 XML 편집기의 설계 및 구현이 이루어져야 할 것이다.

참 고 문 헌

[1] Extensible Markup Language(XML) 1.0, Avalilable http://www.w3.org/TR/REC-xml, 1998.

[2] VoiceXML Forum, Voice eXtensible Markup Lan guage 1.0, March 7, 2000.

[3] William J. Pardi. XML in Action: Web Technology (Microsoft Press), pp. 31-71, pp. 137-159. 1999.

[4] A. D. N. Edwards. Visual Programming Langauges: the Next Generation?. ACM SIGPLAN NotIces. Vol. 23, No.4, pp. 43-50.

[5] Aho, A., Sethi, R. and Ullman, J., Compilers: Principles, Techniques and Tools, Addison-Wesley, 1996.

[6] Thosmas W. Reps, Tim Teitelbaum, The SynthesIzer Generator: A System for Constructing Language-Based Editors, Springer-Verlag 1988.

[7] CLIP! XML Editor v1.5 Avalilable http://xml.t2000.co.kr

[8] XML Pro v2.0 Avalilable http://www.vervet.com/product-index.html

[9] EXml v1.2 Avalilable http://www.cuesoft.com/products/exml.asp

[10] Anthony Karrer and Walt Scacchi, Requirements for an extensible object-oriented tree/graph editor, Proceedings of the the third annual ACM SIGGRAPH symposium on User interface software and technology, pp. 84-91. 1990.

◐ 저 자 소 개 ◑

김 영 철(Kim Young-Chul)
1990년 한남대학교 전자계산학과 졸업(학사)
1996년 숭실대학교 전자계산학과 석사
2003년 숭실대학교 컴퓨터학과 공학박사
현재 : (주)뉴스텍시스템즈 이사, 명지전문대학 겸임교수
관심분야 : 프로그래밍 언어, 컴파일러, XML, 컴퓨터 통신, 소프트웨어공학
E-mail : yckim@ss.ssu.ac.kr

유 두 규(You Du Kyu)
1984년 2월 : 숭실대학교 전기공학과 학사
2001년 2월 : 숭실대학교 컴퓨터교육과 석사
2005년 2월 : 숭실대학교 대학원 컴퓨터학과 박사
1984년 3월~현재 : 세명 컴퓨터 고등학교 재직 중
관심분야 : 네트워크보안, 정보보안, DB보안, DRM
E-mail : bima@dreamwiz.com