XML 구문지향 편집기의 자동 생성 방안

정회원 유 재 우*, 박 호 병**, 조 용 윤***

An Automatic Generation Method of XML Syntax-Directed Editor

Chae-Woo Yoo*, Ho-Byung Park**, Yong-Yoon Cho*** *Regular Members*

요 약

XML은 다양한 분야에서 널리 사용되고 있지만, 일반 사용자가 XML 문서를 작성하기란 아직 많은 어려움이 있다. 본 논문에서는 일반 사용자도 손쉽게 XML 문서를 작성할 수 있는 XML 구문지향 편집기를 소개하고, 구문지향 편집기의 내부 자료구조인 추상구문을 정의하고, 정의된 추상구문 규칙으로 문서를 편집하기 위한 편집기의 구성요소를 설명한다. 또한 DTD로부터 추상구문 규칙을 자동 생성하는 방법을 제시함으로써 더욱 빠르고 정확하게 XML 구문지향 편집기를 생성하는 방안을 제안한다. 추상구문의 구조와 구문지향 편집기의 작성 절차를 통하여 더욱 용이하게 XML 구문지향 편집기 생성이 가능하다.

Key Words : XML, DTD, Syntax-Drected, Editor, Abstract Syntax.

ABSTRACT

While XML is employed in a variety of fields, editing XML document is still hard for the beginners and ordinary individuals. In this paper, we present a syntax-directed editor which is designed to provide unprofessional XML users with easy guides of using XML document. Along with the definition, abstract syntax (data structure of syntax-directed editor) would be explicitly defined. Components of the editor will be projected according to the projected definition of the abstract syntax rule of this paper. Moreover we show that the automatic generation of the abstract syntax rules coming from DTD would enhance the use of XML syntax-directed editor in faster and more precise ways. It could be easier to generate XML syntax-directed editor through a structure of abstract syntax and standard procedure of manufacturing syntax-directed editor.

Ⅰ. 서 론

XML은 데이터를 구조적으로 표현하기 위한 메타 언어로써, 데이터 교환을 위한 중요한 수단으로 사용되고 있으며 컴퓨터 분야 뿐만 아니라 더욱 많은 분야에서 사용되고 있다. 다양한 분야에서 XML을 사용함에 따라 전문가는 물론 일반 사용자들도 XML 문서를 작성해야 하는 필요성이 점차 늘고 있다.

XML은 DTD를 통하여 문서의 구조를 정의하고, 정의된 구조에 따라서 문서를 작성해야 유효한 (valid) 문서가 된다. 유효한 XML 문서를 작성하기 위해서는 DTD의 구조를 정확히 이해하여야 한다. 또한 시작 태그와 끝 태그를 반드시 기술하여야 하고, 시작 태그와 끝 태그는 서로 중첩되지 않아야 하는 등의 정형화(well-formedness) 규칙에 따라야 한다[1]. 따라서 일반 사용자들이 유효한 XML 문서를 작성하기란 쉬운 일이 아니다.

구문지향 편집기는 문법에 익숙하지 않은 사용자

* 숭실대학교 컴퓨터학부 교수 (cwyoo@comp.ssu.ac.kr), ** 숭실대학교 대학원 컴퓨터학과 (r5me@ss.ssu.ac.kr)
*** 숭실대학교 대학원 컴퓨터학과 (yycho@ss.ssu.ac.kr)
논문번호 : KICS2005-03-133, 접수일자 : 2005년 3월 31일
※ 본 연구는 숭실대학교 교내 연구비 지원으로 수행되었습니다.

도 쉽게 문서를 작성하도록 도와주는 대화식 편집기로서, 일반적인 편집기와 컴파일러의 구문 분석 기능을 동시에 가진다. 구문지향 편집기[2]를 이용하면 문법이나 문서의 규칙에 치우치지 않고 문서의 논리적인 설계에 집중할 수 있으며, 항상 올바른 문서를 작성할 수 있다.

본 논문에서는 XML 구문지향 편집기의 구조를 설계하고, 편집기의 내부정보로 사용되는 추상구문[8]에 대하여 소개한다. DTD에서 문서의 규칙 정보를 수집하여 편집기에서 사용되는 추상구문을 생성하는 과정을 자동화함으로써 DTD의 입력만으로 XML 구문지향 편집기를 자동 생성하는 방법을 제시한다.

Ⅱ장에서 XML 구문지향 편집기의 구성에 대하여 기술하고, Ⅲ장에서 DTD를 분석하여 추상구문을 생성하는 규칙을 소개한다. Ⅳ장에서 예제를 통하여 실험을 하고, 마지막으로 Ⅴ장에서 결론 및 향후 연구를 밝힌다.

Ⅱ. XML 구문지향 편집기의 구성

2.1 XML 구문지향 편집기

XML 구문지향 편집기는 현재의 편집 위치에서 편집 가능한 작업을 사용자에게 알려주고, 사용자의 선택에 따라서 문서의 작성이 이루어지는 대화식 편집기이다[2]. XML 구문지향 편집기는 문서를 작성하기 위하여 사용자에게 템플릿을 제공한다. 템플릿은 기본 문서의 구조를 표현하고, 사용자의 입력이 필요한 곳에 place holder를 표시한다. place holder를 통하여 사용자는 문서의 구조를 갱신하고, 추가하면서 점진적으로 문서를 작성하게 된다. XML 구문지향 편집기를 사용하면 문서를 타이핑하는 노력이 감소되고, 이로 인해 결국 타이핑의 에러를 최소화 할 수 있다. 또한 생성된 문서는 문법적으로 항상 유효하다. 따라서 구문적인 오류를 검사하거나 구문 오류를 수정할 필요가 없다. XML 구문지향 편집기는 그림 1과 같이 트리 기반 편집기와 텍스트 기반 편집기로 구성된다. 트리 기반 편집기는 문서의 구조를 트리 형식으로 표현하여 사용자가 문서 전체의 구조를 쉽게 파악하면서 편집이 가능하다. 텍스트 기반 편집기는 내용을 중심으로 하는 편집 환경을 제공한다. 문서의 작성은 사용자의 키보드와 마우스 조작으로 편집기 내부에서 사용하는 추상구문 규칙에 따라서 문서 트리를 점진적으로 생성하여 이루어진다. 두 편집기는 문서 작성 중 사용 가능한 요소들을 선택할 수 있게 해주

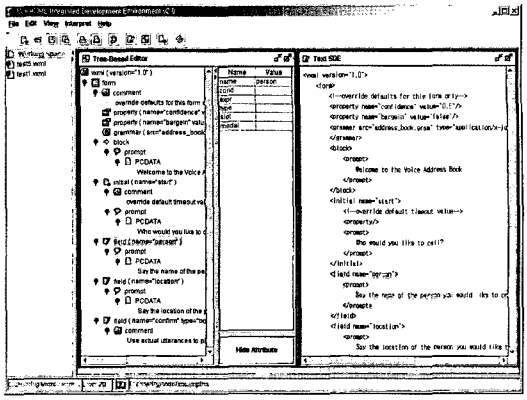


그림 1. XML 구문 지향 편집기

어 사용자가 문서 작성 시 잘못된 요소들을 선택하는 경우를 배제시킨다. 또한 두 개의 편집환경은 상호보완적으로 사용된다. 즉 내용중심의 텍스트 기반 편집은 트리 기반편집기에서도 즉시 반영되고, 역으로 트리 기반 편집기를 이용할 경우에도 텍스트 기반 편집기에 즉시 반영되어 문서 내용을 쉽게 파악할 수 있게 해준다.

2.2 구문지향 편집기의 구성

XML 구문지향 편집기는 추상구문트리와 추상구문 규칙, AST 조작기, 역파서, 이벤트 핸들러, 사용자 인터페이스 등으로 구성된다. 추상구문트리(Abstract Syntax Tree, AST)[3,4]는 XML 문서를 추상화하여 트리 구조로 표현한 문서트리로서 XML 문서에 대한 모든 정보를 포함한다. AST 조작기는 AST를 조작하는 모듈로서, 추상구문 규칙 참조하여 AST에 노드를 추가, 삭제, 변경하는 작업을 수행한다. 추상구문은 구문지향 편집기에서 사용되는 규칙 정보이다. AST의 생성 규칙과 화면 출력 규칙으로 이루어진다. AST 생성 규칙은 AST 조작기에 의해 참조되고, 출력 규칙은 역파서에 의해서 트리 기반 편집기와 텍스트 기반 편집기로 해석되어 진다. 이벤트 핸들러는 사용자의 이벤트를 해석하여 AST 조작기에 요청해서 AST를 변경하는 역할을 한다. XML 구문지향 편집기의 구성은 그림 2와 같다.

추상구문은 AST에 대한 생성 규칙과 AST를 화면에 출력하기 위한 출력 규칙으로 구성된다. 생성 규칙은 $x_0 : op(x_1 x_2 \ldots x_k);$으로 표현한다. op는 생성 규칙을 식별하기 위한 연산자 이름이고, x_i는 문법 기호를 나타낸다. 생성 규칙은 연산자 이름을 제외하면 문맥자유문법(context-free grammar)와 같다. x_0는 문맥자유문법 생성규칙의 좌측(LHS)이고,

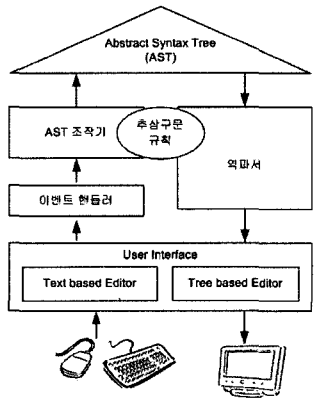


그림 2. 구문지향 편집기의 구성

$x_1x_2...x_k$는 생성규칙의 우측(RHS)에 해당한다. 연산자의 이름을 통하여 동일한 RHS를 갖는 생성규칙을 유일하게 식별할 수 있다[6,7]. 추상구문의 생성규칙은 문맥자유문법과 유사하기 때문에 문맥자유문법으로 표현할 수 있는 문법은 추상구문으로 표현 가능하다. 생성 규칙에서 연산자의 이름은 AST의 노드에 해당된다. AST의 경로(path)는 생성 규칙의 RHS에 대응된다. 또한 확장BNF(EBNF)에서 사용되는 '+', '*', '?' 의 반복 제한자들은 생성 규칙에 속성을 추가함으로써 표현한다. 속성은 'list'와 'optional'이 있다. 속성은 추상구문트리의 생성 방법에 영향을 미친다. 반복 제한자에 대한 추상구문 생성 규칙과 간단한 추상구문트리 변경과정은 표 1과 같이 표현된다.

1번 이상의 반복을 표현하는 '+' 반복 제한자는 무항 연산자(nullary operator)와 우측 재귀(right recursive)하는 이항 연산자(binary operator)로 표현하고, 'list' 속성을 추가한다.

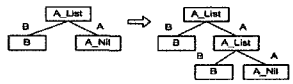
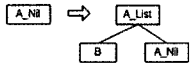
'list' 속성을 가지는 추상구문은 반드시 이항 연산자로 시작하여 무항 연산자로 종결되어야 하고, 무항 연산자를 이항 연산자로 변경하면서 추상구문트리를 구성한다. '*' 반복 제한자는 '+'의 경우와 같은 추상구문 표현을 사용하여 반복을 표현한다. 그러나 'list'속성 대신 'optional' 속성을 사용하여 무항 연산자로 시작하여 무항 연산자를 변경하면서 추상구문트리를 구성한다. '?' 반복 제한자의 경우는 '*' 반복 제한자처럼 'optional' 속성을 이용하여 무항 연산자와 단항 연산자(unary operator)로 표현한다. 무항 연산자로 초기화 된 후, 무항 연산자를 사용자의 선택에 따라서 단항 연산자로 변경하면서 추상구문트리를 구성한다. 추상구문에 속성을 추가하여 추상구문트리를 정의함으로써 다른 구조의 추상구문트리와 일관된 방법으로 해석이 가능하다.

출력 규칙은 추상구문트리를 화면에 출력하기 위한 규칙 정보로서 텍스트 출력 규칙과 트리 출력 규칙으로 구성된다. 출력될 내용은 이중 따옴표를 사용하여 표현하고, 제어문자로 탭문자, 역탭문자를 사용한다. 또한 RHS의 출력 규칙을 적용하기 위하여 '@'기호를 사용한다.

역파서는 추상구문트리에 대하여 정의된 출력 규칙을 해석하여 화면에 출력하는 역할을 한다. 텍스트 출력 규칙은 이중 따옴표로 정의된 내용과 제어문자를 화면에 출력하고 '@' 기호를 만나면 대응되는 RHS에 대한 출력 규칙을 적용한다. 트리 출력 규칙 역시 텍스트 출력 규칙의 경우와 거의 같은 방법으로 적용이 가능하나 탭문자는 이후의 노드가 자식노드임을 나타내고, 역탭문자는 다시 부모노드의 수준으로 다음 노드가 나타남을 의미한다. 이중 따옴표로 정의된 내용이 트리의 노드명으로 나타난다.

구문지향 편집기는 편집기의 정보로서 추상구문 규칙을 이용한다. 추상구문 규칙에 따라서 다양한

표 1. EBNF의 반복제한자와 추상구문트리의 정의

EBNF	추상구문 생성 규칙	추상구문트리 변경과정
A : B+ ;	list A ; A: A_Nil() | A_List (B A) ;	 초기화(1번발생) 노드추가(2번발생)
A : B* ;	optional A ; A: A_Nil() | A_List (B A) ;	 초기화(0번발생) 노드추가(1번발생)
A : B? ;	optional A ; A: A_Nil ( ) | A_B (B) ;	 초기화(0번발생) 노드변경(1번발생)

구문지향 편집기의 생성이 가능하다. 따라서 XML DTD를 추상구문 규칙으로 표현하면 손쉽게 XML에 대한 구문지향 편집기를 생성할 수 있다. 그러나 DTD를 분석하여 추상구문 규칙을 생성하는 작업을 위해서는 DTD에 대한 이해와 추상구문 규칙에 대한 완전한 이해가 필요하다. 본 논문에서는 DTD를 분석하여 추상구문 규칙을 생성하는 과정을 자동화하여 손쉽게 XML 구문지향 편집기를 얻을 수 있다.

Ⅲ. DTD의 분석과 추상구문 규칙

XML 문서는 하나의 루트 원소와 여러 개의 원소들이 내포된 계층 구조를 갖고, 이러한 원소들은 문자 데이터나 자식 원소 혹은 문자 데이터와 자식 원소의 혼합형을 포함한다. DTD는 XML에서 사용하는 데이터의 형식을 정의하는 수단으로 XML 문서의 구조를 정의한다. DTD는 확장된 문맥자유문법(extended context-free grammar)으로 구성된다. DTD에서 원소는 하나 이상의 EMPTY, ANY, 문자 데이터, 하위 원소나 하위 그룹으로 정의된다. 하위 원소나 하위 그룹으로 정의된 원소는 비단말로 구성되며, 문자 데이터나 EMPTY, ANY 등으로 정의된 원소는 단말로 구성된다. 또한 그룹과 하위 그룹, 하위 원소들은 순차적이거나 선택적으로 표현된다. 그룹과 하위 그룹, 하위 원소들은 '?', '*', '+' 등의 반복 제한자를 통해서 제한할 수 있다[1]. DTD가 문맥자유문법과 반복 제한자를 통하여 표현되므로 DTD를 추상구문 생성규칙으로 표현하는 것이 가능하다.

DTD에서 추상구문을 자동 생성하는 과정은 그림 3과 같다. DTD 파서[10]를 통하여 DTD의 유효성을 검사한 후 원소의 정의에 따라 파스트리를 생성한다. 생성된 DTD 파스트리를 구문 지향 편집기에서 사용될 추상구문 규칙으로 변환한다.

DTD 파스트리를 구성하는 노드는 내용 모델을 추상화한 원소 노드와 PCDATA 노드, EMPTY 노드, ANY 노드 등의 단말 노드와 연산자 노드인 SEQ 노드, OR 노드, 그리고 반복 제한 노드인 PLUS 노드와 STAR 노드, OPTIONAL 노드의 비단말 노드로 구성된다.

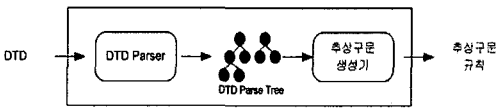


그림 3. 추상 구문 생성 과정

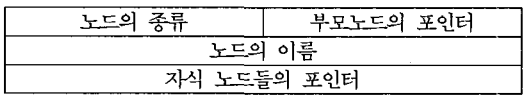


그림 4. DTD 파스트리의 노드 구조

파스트리는 여러 자식을 가질 수 있는 n-ary 구조이다. DTD 원소 정의에서 반복 제한자노드를 생성하고, 반복 제한자에 의해서 영향을 받는 그룹은 반복 제한자 노드의 자식노드가 된다. 트리의 단말노드는 원소노드와 PCDATA노드, EMPTY노드, ANY 노드 등으로 구성된다. 생성된 파스트리는 DTD의 원소의 정의와 같은 의미를 갖는다. DTD 파서를 통하여 파스 트리를 구성하는 알고리즘은 표 2와 같다.

다음은 간단한 메모용 DTD이다.

```
<!ELEMENT memo (sender, receiver,
                content)>
<!ELEMENT content      (#PCDATA)>
<!ELEMENT nickname     (#PCDATA)>
<!ELEMENT firstname    (#PCDATA)>
<!ELEMENT lastname     (#PCDATA)>
<!ELEMENT sender       (person)>
<!ELEMENT receiver     (person)+>
<!ELEMENT person       (nickname |
                (firstname?, lastname))>
```

메모 DTD를 표 2의 알고리즘을 적용하여 생성된 파스트리는 다음의 그림 5와 같다.

원소의 정의별로 파스트리가 생성되며, 메모 DTD는 8개의 파스트리로 구성된다. 생성된 파스트리는 추상구문 생성기의 입력을 통하여 추상구문 규칙을 생성하게 된다. 파스 트리를 입력받아 추상

표 2. DTD 파스트리 생성 알고리즘

```
buildParseTree(DTDItem item, Node parent)
  Node currentNode = parent;
  if (item.cardinal == OPTIONAL 혹은 ZEROMANY 혹은 ONEMANY) {
    Node node = OPTIONAL노드 혹은 STAR노드 혹은 PLUS노드 생성
    node를 currentNode의 자식으로 추가
    currentNode = node }
  switch(item) {
    case ANY:    ANY노드를 생성하여 currentNode의 자식으로 추가
    case EMPTY: EMPTY노드를 생성하여 currentNode의 자식으로 추가
    case ELEMENT: ELEMENT노드를 생성하여 currentNode의 자식으로 추가
    case CHOICE: OR노드를 생성하여 currentNode의 자식으로 추가
          currentNode = OR노드
          Foreach(OR노드의 자식 items) buildParseTree(items, currentNode)
    case SEQUENCE: SEQ노드를 생성하여 currentNode의 자식으로 추가
          currentNode = SEQ노드
          Foreach(SEQ노드의 자식 items) buildParseTree(items,
          currentNode)
    case MIXED: OR노드를 생성하여 currentNode의 자식으로 추가 ;
          currentNode = OR노드
          Foreach(OR노드의 자식 items)
              buildParseTree(items, currentNode)
    case PCDATA: PCDATA노드를 생성하여 currentNode의 자식으로 추가 }
```

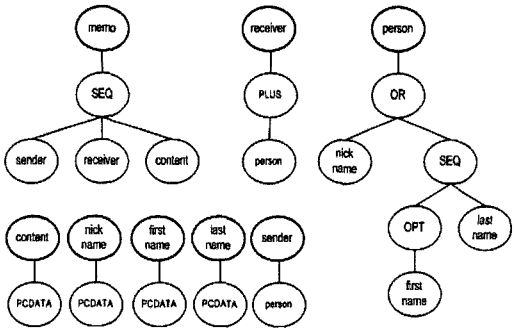


그림 5. 메모 DTD에 대한 파스트리

표 3. 추상구문 규칙 생성 알고리즘

```
generateAST (Node parent, 추상구문 규칙의 현재 행 row) {
   switch (parent노드의 종류)
      case ELEMENT:
         row의 LHS = parent노드의 이름 ; row의 OPNAME =
         자식노드의 경로명
         generateSubAST(자식노드)
   case PLUS : CASE STAR :
      if (parent노드의 종류 == PLUS) 'list'속성 추가
      else 'optional'속성 추가
      row의 LHS = parent노드의 경로명
      row의 OPNAME = 자식노드의 경로명+"NIL"
      row의 RHS = NULL ; newRow = 추상구문 규칙의 새로운 행
      newRow의 OPNAME = 자식노드의 경로명+"LIST"
      generateSubAST(자식노드) ; newRow의 RHS에 row의 LHS 추가
   case OPTIONAL :
      'optional'속성 추가 ; row의 LHS = parent노드의 경로명
      row의 OPNAME = 자식노드의 경로명+"NIL"
      row의 RHS = NULL ; newRow = 추상구문 규칙의 새로운 행
      newRow의 OPNAME = 자식노드의 경로명 ; generateSubAST
      (자식노드)
   case OR:
      row의 LHS = parent노드의 경로명
      row의 OPNAME = parent노드의 경로명+"NIL"
      row의 RHS = NULL
      foreach (parent노드의 자식노드 child){ curRow = 추상구문 규칙의
      새로운 행
         curRow의 OPNAME = child노드의 경로명
         generateSubAST(child) }
   case SEQ:
      row의 LHS = parent노드의 경로명
      row의 OPNAME = parent노드의 첫번째 자식 경로명
      foreach (parent노드의 자식노드 child)  generateSubAST(child)
}
generateSubAST (Node child) {
   if (child노드 == 단말노드)
      if(child노드의 종류 == EMPTY) row의 RHS = NULL
      else row의 RHS = child노드의 이름
   else    row의 RHS = child노드의 경로명
      generateAST(child, 추상구문 규칙의 새로운 행)
}
```

구문 규칙을 생성하는 알고리즘은 표 3과 같다. 파스트리의 루트노드를 현재 노드로 하여, 노드의 종류에 따라서 추상구문 규칙을 완성한다. 현재 노드의 종류가 원소노드인 경우는 하나의 자식 노드를 갖고, LHS는 현재 노드의 이름이 된다. 연산자 이름은 유일한 이름을 부여하기 위하여 자식 노드의 경로명을 취한다. RHS는 자식 노드가 단말 노드인 경우에는 단말 노드의 이름이 되고, 비단말 노드인 경우에는 새로운 비단말 기호를 생성하고 새로운 비단말을 자식노드의 경로명으로 부여한다. 새로운 비단말이 생성되었으므로 자식노드를 현재 노드로 하여 재귀적으로 호출하여 단말 노드로 끝날 때 까지 반복한다. PLUS노드는 'list' 속성을 갖고, STAR 노드는 'optional' 속성을 갖는다. PLUS노드와 STAR 노드는 추상구문으로 표현할 때 무항 연산자와 이항 연산자를 이용하여 우측재귀하도록 표현한다. 'list'속성을 갖는 추상구문 규칙의 적용은 항상 이항 연산자가 먼저 사용되고, 그 이후에 무항 연산자가 사용된다. OPTIONAL노드는 사용자의 편집위치에 따라서 무항 연산자 노드의 화면 출력여부가 결정되므로 'optional'속성을 갖는다. OPTIONAL 노드는 하나의 자식 노드를 갖고 있고, 자식노드가 0번 혹은 1번 발생하므로 무항 연산자와 단항 연산자로 구성하고, 자식 노드가 단말노드이면 RHS로 자식 노드의 이름을 사용하고, 비단말 노드이면 자식 노드의 경로명을 사용한 후, 비단말 노드에 대하여 재귀적으로 생성 알고리즘을 적용한다. OR노드는 추상구문트리를 사용자가 선택하여 변경할 수 있도록 무항 연산자를 추가하고, n개의 자식들에 대해서 n개의 추상구문 규칙을 생성한다. SEQ노드는 n개의 자식들에 대하여 n개의 RHS를 갖는 하나의 추상구문 규칙을 생성한다. 마지막으로 추상구문 규칙의 시작을 표시하기 위하여 'root' 생성 규칙을 추가한다. AST를 화면에 텍스트 형식과 트리 형식으로 출력하기 위하여 두 가지의 출력 규칙을 정의한다. 추상구문 규칙의 출력 형식 지정은 표 4와 같다.

LHS가 원소인 경우에 텍스트 출력 규칙은 원소의 이름을 태그 형식으로 표현하고, RHS의 규칙을 적용하기 위하여 '@' 기호를 RHS의 개수만큼 지정한다. 원소의 태그 형식 표현과 RHS는 부모-자식 관계에 해당되므로 들여쓰기와 내어쓰기를 위하여

표 4. AST의 출력 규칙

```
if (LHS == 원소)
   if (RHS == NULL) 텍스트 출력 규칙에 "<원소이름/>" 추가
      트리 출력 규칙에 "원소이름" 추가
   else 텍스트 출력 규칙에 "<원소이름>\t" 추가 ; 트리 출력 규칙에 "원소이름\t" 추가
      RHS처리함수호출
      텍스트"\b</원소이름>" 추가 ; 트리 출력 규칙에 "\b" 추가
else RHS처리함수호출
RHS처리() {
   if (RHS == NULL)
      if ('list' 혹은 'optional' 속성) 텍스트 출력 규칙에 "<!- expand -->" 추가
         트리 출력 규칙에 "<expand>"추가
      else-if (LHS == PCDATA) 텍스트 와 트리 출력 규칙에 TEXT추가
      else 텍스트 출력 규칙에 "<!- select -->" 추가
         트리 출력 규칙에 "<select>" 추가
   else
      foreach (RHS의 각 rhs) {
         if (rhs == PCDATA) 텍스트 와 트리 출력 규칙에 TEXT 추가
         else 텍스트 와 트리 출력 규칙에 @ 추가 }
}
```

표 5. 메모 DTD에 대한 추상구문 규칙

```
root memo;
memo : memo_SEQ (memo_SEQ)
   ["<memo>\t" @ "\b</memo>"] {"memo\t"@"\b"};
memo_SEQ : memo_SEQ_sender (sender receiver content) [@@@] {@@@};
content : content_PCDATA (PCDATA)
   ["<content>\t" @ "\b</content>"]{"content\t"@"\b"};
nickname : nickname_PCDATA (PCDATA)
   ["<nickname>\t" @ "\b</nickname>"]{"nickname\t"@"\b"};
firstname : firstname_PCDATA (PCDATA)
   ["<firstname>\t" @ "\b</firstname>"]{"firstname\t"@"\b"};
lastname : lastname_PCDATA (PCDATA)
   ["<lastname>\t" @ "\b</lastname>"]{"lastname\t"@"\b"};
sender : sender_person (person)
   ["<sender>\t"@"\b</sender>"] {"sender\t"@"\b"};
receiver : receiver_PLUS (receiver_PLUS)
   ["<receiver>\t"@"\b</receiver>"] {"receiver\t"@"\b"};
list receiver_PLUS;
receiver_PLUS : receiver_PLUS_NIL() {"<!- expand -->"} {"<expand>"}
                | receiver_PLUS_LIST(person receiver_PLUS) [@@] {@@};
person : person_OR(person_OR)
   ["<person>\t"@"\b</person>"] {"person\t"@"\b"};
person_OR : person_OR_NIL() {"<!- select -->"} {"<select>"}
                | person_OR_nickname(nickname) [@] {@}
                | person_OR_SEQ (person_OR_SEQ) [@] {@};
person_OR_SEQ : person_OR_SEQ_OPT (person_OR_SEQ_OPT lastname)
   [@@] {@@};
optional person_OR_SEQ_OPT;
person_OR_SEQ_OPT : person_OR_SEQ_OPT_NIL()
   ["<!- expand -->"] {"<expand>"}
                | person_OR_SEQ_OPT_firstname(firstname) [@] {@};
PCDATA : PCDATA ( ) [TEXT] {TEXT};
```

탭문자와 역탭문자를 사용한다. 텍스트 출력 규칙에서 '@'는 줄띄기를 적용하여 출력한다. 트리 출력 규칙은 원소의 이름을 노드의 이름으로 사용하고, 부모-자식 간의 관계를 탭문자와 역탭문자로 제어한다. RHS가 NULL인 경우, 만일 LHS가 원소이면 태그의 축약형으로 표현하고, 'list'나 'optional'속성을 갖으면 AST를 확장할 수 있도록 적절한 메시지를 지정한다. 여기에서는 'expand'라는 메시지를 사용한다. 그리고 속성을 갖지 않은 경우에는 AST를 변경할 수 있도록 'select'라는 메시지를 이용하여 AST에서 다른 노드를 선택하도록 한다.

추상구문 생성 규칙을 적용하여 생성된 예제 DTD에 대한 추상구문 규칙은 표 5와 같다.

XML 구문지향 편집기는 DTD를 자동으로 추상구문 규칙으로 변환하고, 변환된 추상구문 규칙을 인식하여 항상 유효한 XML 문서를 생성할 수 있다.

다양한 DTD에 대한 입력만으로 변환되는 추상구문을 자동으로 생성함으로써 손쉽게 XML 구문지향 편집기의 생성이 가능하다.

Ⅳ. 실 험

XML 구문지향 편집기를 통한 유효한 XML 문서의 작성 과정을 알아본다. 예제는 메모 DTD로부터 생성된 추상구문 규칙을 이용한다. 추상구문 규칙을 통하여 초기화된 추상구문트리의 모양과 이에 대한 편집기의 모습은 그림 6과 같다. 추상구문트리에서 단말 노드들에 대하여 점진적으로 노드를 추가하면서 문서를 작성하게 된다. 추상구문 트리의 단말 노드에 대하여 메뉴를 통하여 변경 가능한 노드를 추천하고 사용자의 선택으로 노드를 추가하게 된다.

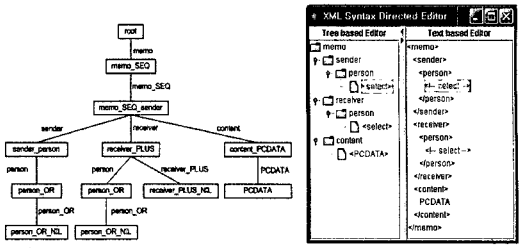


그림 6. 초기화된 추상구문트리와 구문지향 편집기

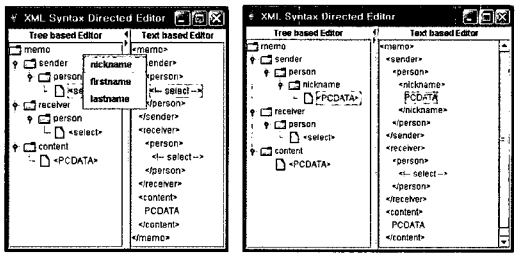


그림 7. 노드의 선택과 변경

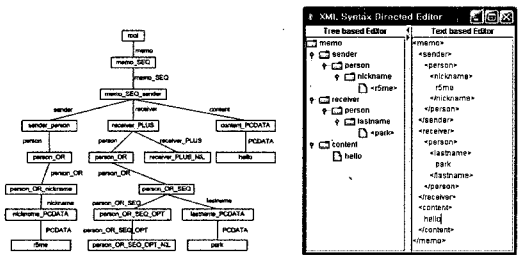


그림 8. 완성된 추상구문트리와 구문지향 편집기

그림 7은 사용자의 현재 위치에서 선택 가능한 원소를 추상구문 규칙으로부터 얻고, 사용자가 선택에 의하여 추상구문트리가 변경되고 그 결과를 역파서에 의해서 다시 화면에 출력한 결과이다. 이와 같이 점진적 과정을 거쳐서 생성된 최종결과는 그림 8과 같다.

위의 예를 통하여 보편적으로 사용할 수 있는 간단한 DTD에서 문서의 생성규칙 정보를 자동으로 추출하여 추상구문 규칙을 생성하고, 생성된 추상구문 규칙을 통하여 문서 작성하는 과정을 보였다. 본 연구에서 제시한 알고리즘과 자료구조, 처리 절차를 통하여 행정자치부의 전자민원서식 표준 DTD로 화

재증명원신청서 DTD[9]와 VXML[11]과 XTM[12]등의 XML 응용 프로그램에 대하여 적용하여 실험하였으며, 원소의 속성을 제외한 나머지 부분에 대하여서는 유효한 XML 문서를 작성할 수 있었다.

V. 결 론

XML은 표준으로 제정된 이후에 컴퓨터뿐만 아니라 산업 전 분야에서 데이터를 표현하기 위해서 사용되고 있다. 이제 XML 문서는 전문가가 아닌 일반인들도 작성해야 할 필요성이 커지고 있다. 그러나 사용자가 XML 문법에 맞게 문서를 작성하는 것이 쉽지 않다. 이러한 어려움을 해결하기 위하여 XML 구문지향 편집기가 사용되고 있다. 본 논문에서는 DTD의 입력으로 XML 구문지향 편집기를 자동 생성하기 위한 절차와 필요한 자료구조를 정의하였다. 편집기에서 사용하는 자료표현인 추상구문을 소개하고, 추상구문트리를 다양한 형태로 출력하는 다중 뷰를 통하여 다양한 관점으로 문서를 작성할 수 있도록 하였다. 또한 DTD를 분석하여 추상구문 정의를 생성하는 과정을 자동화함으로써 손쉽게 추상구문을 생성할 수 있다. XML 편집기를 작성하기 위한 표준화된 절차와 자료구조를 통하여 다양한 XML 구문지향 편집기를 작성할 때 유용하게 이용할 수 있을 것으로 생각한다.

현재 DTD에서 추상구문의 정의와 자동생성은 원소의 정의를 중심으로 구현되었으며, 향후 원소의 속성 정의에 대한 부분을 지원하는 방법을 고려할 것이다.

참 고 문 헌

[1] Extensible Markup Language(XML) 1.0, available at http://www.w3.org/TR/REC- xml, 2004.

[2] A. A. Khwaja and J. E. Urban, Syntax- directed Editing Environments: Issues and Features", ACM-SAC, pp. 230-237, 1993.

[3] F. Arefi, C. E. Hughes, and D. A. workman, "Automatic Generating Visual Syntax-Directed Editors", Communications of the ACM, Vol. 33, No. 3, pp. 349-360, March 1990.

[4] A. V. Aho, R. Sethi and J. D. Ulman, Compilers: Principles, Techniques and Tools, Addison-Wesley, 1996.

[5] L. Allision, "Syntax Directed Program Editing", Software Practice and Experience, Vol. 13, No. 5, pp. 453-465, May 1983.

[6] T. W. Reps and T. Teitelbaum, The Synthesizer Generator: A System for Constructing Language-Based Editors, Springer-Verlag, 1989.

[7] GrammaTech, The Synthesizer Generator: Specifying an Editing Environment, available at http://www.grammatech.com/research/SGspecifying/specifying.html

[8] 신경희, 최종명, 유재우, "XML 문서 편집을 위한 추상문법", 정보과학회논문지, 제30권, 제3호, pp. 268-277, April, 2003.

[9] 행정자치부, "행정기관의 전자민원서식 표준(안)", available at http://www.mogaha.go.kr/warp/webapp/board/notice/list?bid=106, Dec 2004.

[10] Wutka, JAVA DTD Parser, available at http://www.wutka.com/dtdparser.html

[11] VoiceXML Recommendation, available at http://www.w3.org/TR/2004/REC-voicexml20-200403 16.

[12] TopicMaps.org Authoring Group, available at http://www.topicmaps.org.

유 재 우(Chae-Woo Yoo) 정회원

1976년 숭실대학교 전자계산학과(공학사)

1978년~1987년 한국과학기술원(공학석사, 박사)

1986년~1987년, 1996년~1997년 Cornell Univ. Univ. of Pittsburgh Visiting Scientist

1983년~현재 숭실대학교 컴퓨터학부 교수

<관심분야> 컴파일러, 프로그래밍 환경, HCI, XML

박 호 병(Ho-Byung Park) 정회원

1999년 숭실대학교 전자계산학과(공학사)

2002년 숭실대학교 대학원 컴퓨터학과(공학석사)

2002년 9월~현재 숭실대학교 대학원 컴퓨터학과 박사과정

<관심분야> 컴파일러, 프로그래밍 환경, XML

조 용 윤(Yong-Yoon Cho) 정회원

1995년 인천대학교 전자계산학과(공학사)

1998년 숭실대학교 대학원 전자계산학과(공학석사)

1999년~현재 숭실대학교 대학원 컴퓨터학과 박사과정

<관심분야> 컴파일러, 프로그래밍 환경, XML