# XED: 데이타 중심 XML 문서를 위한 모델 기반의 XML 편집기 생성 도구

## (XED: Model-based XML Editor Generator for Data-Centric XML Documents)

최 종 명 † 유 재 우 ††
(Jong-Myung Choi) (Chae-Woo Yoo)

**요 약** XML은 점차 널리 사용되고 있지만, 일반 사용자가 편집기를 이용해서 XML 문서를 작성하기에는 아직 많은 어려움이 있다. XML 문서 중에서 상당 부분은 정형화된 데이타를 처리하기 위한 데이타 중심 XML 문서이며, 이러한 형태의 문서인 경우에 초보자도 폼(form) 형태의 GUI를 이용해서 쉽게 작성할 수 있다. 본 논문에서는 데이타 중심 XML 문서에 대해서 모델 기반으로 폼 형태의 XML 편집기를 자동적으로 생성할 수 있는 방법과 편집기 생성기인 XED를 소개한다. XML 문서의 DTD는 연속, 선택, 반복의 구조로 이루어져 있으며, 이러한 구조는 DDG (Document Decomposition Graph) 그래프로 표현될 수 있다. XED는 XML의 DDG에 사용자가 프리젠테이션 규칙을 적용함으로써 XML 편집기를 자동적으로 생성할 수 있고, 사용자가 생성된 편집기의 레이아웃과 GUI 속성을 직접 조작을 통해서 변경할 수 있는 방법을 제공한다.

**키워드** : XML, 편집기 생성기, 태스크 모델

***Abstract*** Though XML is widely used, it is still hard for end users to write XML documents. A lot of XML documents are data-centric documents which have formal data format. Even novices can easily write the data-centric XML documents if they use form-based GUIs. In this paper, we introduce a new method for generating form-based XML editor for data-centric XML documents automatically and an XML editor generator called XED. The DTD consists of sequence, choice, and repetition, and this structure can be represented with Document Decomposition Graph(DDG). XED allows users to generate an XML editor by applying the presentation rules to DDG. It also permits users to modify generated editor through changing editor's GUI properties with direct manipulation.

**Key words** : XML, editor generator, task model

## 1. 서 론

XML은 데이타와 문서를 기술하기 위한 표준으로 제정된 이후에 컴퓨터 전 분야에서 사용되고 있으며, 앞으로 더욱 많은 분야에서 사용되게 될 것이다. 많은 분야에서 XML을 사용함에 따라 전문가는 물론 일반 사용자들도 XML 문서를 작성해야하는 필요성이 점차 늘고 있다. 그러나 일반 사용자가 XML 문서를 작성하기 위해서는 XML에 대한 전문적인 지식을 익혀야 하는 어려움이 있다. 예를 들어, 시작 태그와 끝 태그를 반드시 기술하여야 한다든지 혹은 속성 값은 항상 인용 부호를 붙여야 한다는 등의 XML 규칙을 알고 있어야 한다. 또한 작성하려는 XML 문서의 구조와 어휘를 사전에 파악해야 올바른 XML 문서를 작성할 수 있다.

현재 널리 사용되는 XML편집기로는 XMLSpy[1]와 IBM의 Xeena[2]가 있으며, 이러한 XML 편집기는 DTD 기반의 트리 형식으로 문서를 작성하는 트리 편집기 방식을 사용한다. 트리 편집기는 문서의 구조를 트리 형식으로 표현하고, 사용자는 각 트리 노드에 데이타들을 입력한다. 또 다른 형태의 XML 편집기로는 문서의 구조를 이용한 구문 지향 편집기[3]가 있다. 구문 지향 편집기는 XML 문서 구조에 따라 다음에 사용할 수 있는 태그들을 보여주고 사용자가 선택할 수 있도록 한다. 이러한 편집기들은 임의의 DTD가 주어지는 경우에

· 본 연구는 숭실대학교 교내 연구비 지원으로 이루어졌음
† 학생회원 : 숭실대학교 컴퓨터학과
jmchoi@comp.ssu.ac.kr
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
cwyoo@comp.ssu.ac.kr
논문접수 : 2002년 9월 25일
심사완료 : 2003년 6월 24일

DTD에 맞게 문서를 작성할 수 있는 기능들을 제공한다.

트리 기반 편집기와 구문지향 편집기는 범용 XML 문서를 작성할 수 있는 장점이 있지만, 일반 사용자가 사용하기에는 어려운 점이 많다. 첫째로, 상용 XML 편집기들은 너무 많은 기능들을 제공하기 때문에 일반 사용자들이 XML 편집기의 사용법을 익히기에는 상당한 노력을 요구한다. 초보자들에게는 복잡하고 많은 기능들보다는 자주 사용되는 간단한 기능들만 제공되는 인터페이스가 훨씬 효과적이다[4]. 둘째로, XML 문서를 작성하기 위해서 사용자들은 XML 문서의 전체적인 구조, 원소와 속성의 의미에 대한 사전 지식을 필요로 한다. 셋째로, 사용자의 실수로 인해 잘못 작성한 XML 문서인 경우에 문서의 오류를 찾기 위해서 문서를 꼼꼼히 살펴보아야 한다. 대부분의 XML 편집기는 기본적으로 XML 문서의 문법과 유효성을 체크해주는 기능을 제공한다. 그러나 의미적인 체크는 수행하지 못하기 때문에 XML 문서에 오류는 여전히 존재하게 된다. 이러한 어려움들 때문에 일반 사용자가 좀더 쉽게 XML 문서를 작성할 수 있는 편집기가 필요해진다.

XML 문서는 데이타 형태에 따라서 문서 중심(document-centric) XML 문서와 데이타 중심(data-centric) XML 문서로 분류할 수 있다[5]. 문서 중심 형태는 주로 사람이 읽기 위한 문서로서 원소와 #PCDATA가 같이 사용되는 혼합된 내용(mixed contents)을 많이 갖고, 순서가 중요하며, 기계적으로 처리하기 어려운 특징이 있다. 반면에 데이타 중심 문서는 기계적 처리가 쉽고, 문서 형태가 정형화된 특징이 있다. 첫 번째 형태의 문서 예로는 책, 광고 등이 있고, 두 번째 형태로는 상품 주문서, 주식 데이타, 보고서, 과학 데이타 등이 있다.

한편 현재 사무실 환경에서 데이타 입력을 위해서 사용되는 응용프로그램들은 메뉴와 폼(form) 방식으로 사용자가 원하는 내용을 입력하거나 혹은 선택할 수 있는 GUI를 제공한다. 이러한 응용프로그램들에서 사용되는 데이타들은 대부분 데이타 중심 XML 문서로 표현될 수 있다. 폼 기반 편집기를 사용하는 경우에 사용자는 XML 문서가 어떤 구조를 가지고 있으며, 각 데이타 필드의 이름이 무엇이고, 어떤 형태로 저장되는지에 대해 알고 있을 필요가 없다. 이러한 사용자 인터페이스는 배우기 쉽고, 사용하기 편리하며, 효과적으로 문서를 작성할 수 있고, 입력되는 데이타의 오류도 줄여줄 수 있는 장점을 가지고 있다. 예를 들어, J2EE SDK[6]에서 EJB(Enterprise JavaBeans) 응용프로그램을 배포하기 위한 deploytool은 XML 문서를 생성해야 함에도 불구하고, 사용자가 XML 문서의 구조와 내용을 모르면서 문서를 작성할 수 있도록 도와준다. 따라서 일반 사용자

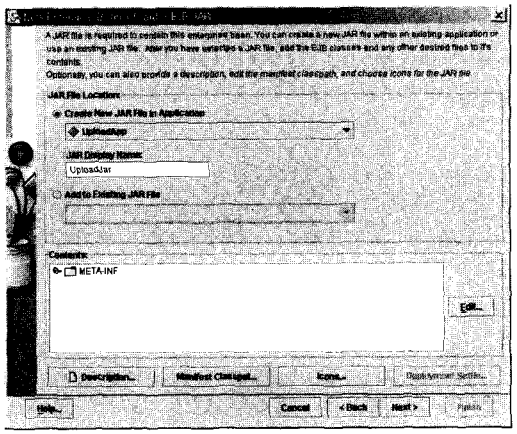


그림 1 J2EE SDK 배치 도구

에게 필요한 XML 편집기는 각 해당 필드의 데이타를 직접 입력하거나 데이타 목록에서 선택할 수 있는 형태의 GUI를 제공함으로써 사용자는 자신이 작성하는 내용이 XML로 저장되는지 혹은 다른 형태로 저장되는지 인식하지 못하면서 데이타를 작성할 수 있어야 한다. 그림 1은 편리한 GUI를 제공함으로써 EJB 응용프로그램 배포를 위한 XML 문서를 작성할 수 있는 J2EE SDK의 deploytool이다.

이러한 필요성을 만족시켜주기 위해서 본 논문에서는 선언적 모델을 기반으로 폼 형태의 XML 편집기를 자동적으로 생성할 수 있는 방법과 XED(XML Editor Designer)라는 편집기 생성기를 소개한다. XED는 XML 문서의 DTD와 스타일 규칙을 이용해서 자동적으로 XML 편집기를 생성한다. XED는 또한 자동적으로 생성된 XML 편집기에서 GUI 컴포넌트들의 레이아웃과 속성 등을 사용자가 커스터마이징할 수 있도록 GUI 디자인 도구의 직접 조작 방식을 지원한다. 직접 조작 방식은 생성된 XML 인터페이스의 스타일과 속성을 사용자가 원하는 형태로 지정할 수 있다.

본 논문은 2장에서 관련 연구를 기술하고, 3장에서 XML에서 태스크 모델과 인터페이스 스타일 규칙을 소개한다. 4장에서는 XML 편집기 생성기를 소개하고, 마지막으로 5장에서 결론 및 향후 연구를 밝힌다.

## 2. 관련 연구

### 2.1 XML EditorMaker

XML EditorMaker[7]는 IBM에서 개발된 XML 편집기 생성기이다. XML EditorMaker는 문서의 DTD가 주어지는 경우에 XML 문서를 입력할 수 있는 편집기를 자동적으로 생성한다. XML EditorMaker가 생성하는 XML 편집기는 자식 원소들을 갖는 부모 원소는 별

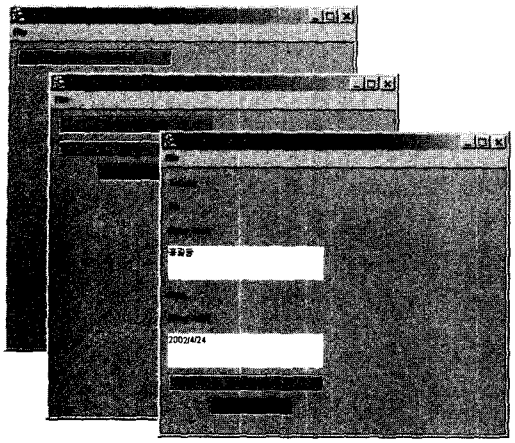

그림 2 XML EditorMaker

도의 창에서 데이타들을 입력하도록 여러 개의 창들로 구성된다. XML EditorMaker는 기존의 XML 트리 편집기나 구문 지향 편집기와는 다르게 XML 문서 구조에 따라 여러 개의 창에 데이타를 입력할 수 있도록 지원한다는 점에서 독창적이다. 그러나 XML EditorMaker는 실세계에서 사용하기에는 여러 가지 단점들을 가지고 있다. 첫째로 XML EditorMaker는 인터페이스 스타일 정보를 전혀 사용하지 않으며, 생성된 XML 편집기의 인터페이스를 커스터마이징 할 수 있는 방법을 제공하지 않는다. 둘째로 XML EditorMaker를 이용해서 생성된 편집기는 데이타를 입력받기 위해서 단순한 텍스트 영역만 제공한다. 데이타는 필요에 따라 텍스트 영역, 리스, 콤보박스, 체크박스 등의 다양한 컴포넌트를 통해서 입력되어야 하는데, 생성된 편집기는 이러한 기능을 제공하지 못한다. 그림 2는 XML EditorMaker를 통해서 생성된 XML 편집기의 모습이다.

### 2.2 Teallach

Teallach[8,9]는 객체지향 데이타베이스를 위한 사용자 인터페이스를 개발하기 위한 MB-UIDE (Model Based User Interface Development Environment)이다. Teallach는 도메인 모델, 태스크 모델, 프리젠테이션 모델을 이용해서 사용자 인터페이스를 자동적으로 생성할 수 있도록 지원한다. Teallach는 각 모델을 지원하기 위해서 도메인 모델 편집기, 프리젠테이션 모델 편집기, 태스크 모델 편집기를 제공하고, 각 모델 편집기들 간에 드래그와 링크를 이용해서 서로 연동할 수 있는 기능을 제공한다.

## 3. 본 론

### 3.1 태스크 모델

사용자 인터페이스 개발에는 많은 비용과 시간이 소비되기 때문에 이것을 효과적으로 개발하기 위한 많은 연구들이 수행되고 있다[10]. 모델 기반 인터페이스 개발은 GUI 인터페이스 개발에 적합하고, 사용자 인터페이스를 자동적으로 생성할 수 있는 장점을 가지고 있다. 모델 기반 인터페이스 개발에서 태스크 모델(task model)은 가장 먼저 작성해야 하는 것으로 알려져 있다[11]. 태스크는 주어진 컨텍스트 상황에서 중요한 상태 변화를 가져오는 사용자의 행위를 의미한다[12]. 태스크 분석은 데이타 흐름도를 이용하는 방법[11], 이벤트를 이용하는 방법[13] 등을 사용하고, 기본적으로는 태스크를 하향식 기능 분석을 통해 서브 태스크로 분할해나가는 방식으로 이루어진다. 태스크의 분할은 서브 태스크가 액션 혹은 객체로 표현될 때까지 반복적으로 수행된다. 태스크 분석이 이루어지면, 최종적으로 액션과 객체들을 얻을 수 있고, 또한 객체들간의 관계(is-member, has-a, is-a-group)와 액션들간의 관계(sequential, parallel, alternative, repetition)도 파악할 수 있다.

본 논문에서는 이벤트를 이용해서 태스크를 분할하는 방법을 사용한다. 이벤트를 통해서 분할된 서브 태스크들은 EDG(Event Decomposition Graph)를 통해서 표현될 수 있다[13]. EDG에서 단말 노드는 완료된 이벤트를 의미한다. 그림 3은 메일을 전송하기 위한 태스크를 EDG를 이용해서 표현한 것이다[11]. EDG는 AND/OR 그래프를 이용한 것이고, 이 그래프는 2가지 형태의 링크를 가지고 있다.

- AND - 모든 자식 노드들이 유효한 경우에 부모 노드가 유효하다.
- OR - 자식 노드들 중에 오직 하나의 노드가 유효

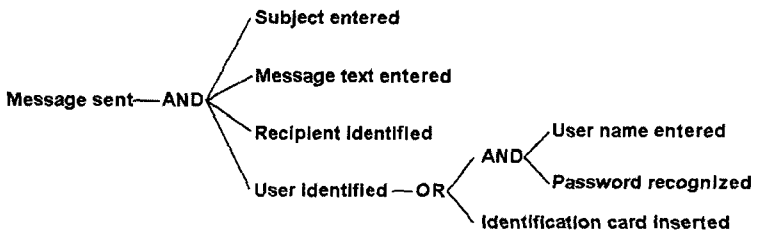


그림 3 메일 전송을 위한 EDG

한 경우에 부모 노드가 유효하다.

메일을 전송하는 태스크는 AND를 이용해서 네 개의 서브 태스크로 분할될 수 있고, 메일 전송 태스크가 유효하기 위해서는 네 개의 서브 태스크들이 모두 유효해야 한다. OR로 분할되는 경우에는 서브 태스크 중에서 하나만 유효한 경우에도 상위 태스크가 유효하다. 메일 전송에서 사용자 인식 서브 태스크는 OR를 이용해서 두개의 서브 태스크로 분할된다.

DTD는 XML 문서의 구조와 어휘를 정의하고, 유효한 XML 문서는 DTD에 맞게 작성되어야 한다. XML 문서는 가장 상위에 루트 노드를 갖는 트리 형식으로 표현되고, DTD의 규칙에 따라 노드들의 관계가 결정된다. 노드들의 관계는 순차, 선택, 반복, 옵션의 형태를 갖는다. 일반적으로 XML 문서 작성은 문서의 구조에 따라 순차적으로 작성하기 때문에 문서 구조를 태스크 모델로 모델링할 수 있다. 즉, XML 문서의 내용을 상단에서 하단으로 작성해나가는 것은 태스크의 흐름으로 이해할 수 있으며, 문서의 각 원소들은 서브 태스크로 파악할 수 있다. XML 문서에 적용되는 태스크 모델은 XML 문서 구조를 분석하게 되고, XML 문서 구조는 이벤트를 이용한 태스크 분석 방법을 적용할 수 있다. 즉, XML 문서의 트리 구조에서 노드들은 "데이타 입력 이벤트"로 표현할 수 있다. 따라서 XML 문서 구조는 EDG와 유사하면서 XML 문서 구조의 특성인 옵션, 0번 이상 반복, 1번 이상 반복 등에 대한 사항들을 추가한 그래프로 표현할 수 있다. 이 그래프를 DDG (Document Decomposition Graph)라고 부르기로 한다.

**정의 1.** DDG

XML 문서의 이벤트 그래프는 DDG = <V, A>이다. V = { $X_1$, $X_2$, ..., $X_n$ }이고, A = { ($X_i$, $X_j$) | $X_i$, $X_j$ ∈ V }이다. DDG의 노드는 데이타 입력 이벤트(DE)와 DTD의 구조(DS)들로 구성되어 있다. 따라서 V = DE ∪ DS로 정의될 수 있다. 또한 데이타 입력 이벤트 DE는 다시 원소 데이타 입력 이벤트(EL)와 속성 데이타 입력 이벤트(AT)로 구성된다. 즉, DE = EL ∪ AT로 정의할 수 있다. EL은 EL = { element | element defined in DTD }로 정의하고, AT는 AT = { attribute | attribute defined in DTD }로 정의할 수 있다. DTD의 구조 DS는 DS = { SEQ, OR, ?, +, * }이다. □

XML 문서의 DTD를 DDG로 표현하는 것은 상대적으로 간단하게 이루어질 수 있다. DTD에서 원소의 정의는 ECFG (Extended Context Free Grammar)로 표현될 수 있다. 따라서, DTD의 원소 정의를 생성 규칙으로 표현하고, 생성 규칙을 통해서 DDG의 서브 트리를 구성할 수 있다.

**정리 1.** SEQ와 OR 그룹

원소의 내용이 자식 원소로 구성된 경우에 자식 원소들은 SEQ 혹은 OR 그룹으로 묶인다. SEQ는 원소 내용이 순차(sequence)로 선언된 경우에 사용되고, OR는 원소 내용이 선택(choice)으로 선언된 경우에 사용된다. 이때 원소 A의 정의는 그룹(SEQ, OR)을 유도하는 생성 규칙과 그룹이 자식 원소를 유도하는 생성 규칙으로 변환될 수 있다. □

예를 들어, 정리 1에 의하면, <!ELEMENT A ( $B_1$ | $B_2$ | .. | $B_n$ )> 형태의 원소 정의는 다음과 같은 두개의 생성 규칙으로 변환될 수 있다.

A -> A_OR - 생성 규칙 1

A_OR -> $B_1$ | $B_2$ | .. | $B_n$ - 생성 규칙 2

생성 규칙의 우측은 DDG에서 좌측 노드의 자식 노드로 표현된다. 따라서 생성 규칙 1과 2는 DDG에서 그림 4와 같은 서브 트리를 구성하게 된다.

**정리 2.** 반복과 옵션

원소와 그룹에 적용되는 반복과 옵션은 DDG에서 원소와 그룹의 부모 노드로 표현된다. □

예를 들어, 원소 정의 <!ELEMENT A ( $B_1$, $B_2$?, ..., $B_n$ )*>는 다음과 같은 생성 규칙으로 변환된다.

A -> A_0_MORE - 생성 규칙 3

A_0_MORE -> A_SEQ - 생성 규칙 4

A_SEQ -> $B_1$, $B_2$_?, ..., $B_n$ - 생성 규칙 5

$B_2$_? -> $B_2$ - 생성 규칙 6

생성 규칙 3-6는 그림 5와 같은 서브 트리를 형성하게 된다.

DDG의 단말 노드들은 실제 데이타가 입력되는 부분들이다. DTD에서 #PCDATA, EMPTY, ANY 내용을 갖는 원소와 속성은 단말 노드로 표현된다. 반면에 DTD에서 자식 노드를 갖는 원소, SEQ, OR, +, *, ?은 DDG에서 중간 노드를 구성하게 된다. SEQ는 EDG의 AND와 동일한 의미이고, OR는 EDG의 OR와 동일한 의미이다. +는 1번 이상 반복, *는 0번 이상 반복을 의미하고, ?는 옵션을 의미한다. DDG는 EDG와는 다르게 SEQ, OR 등을 아크가 아닌 노드로 표현한다. 중간 노드들은 이후에 프리젠테이션 규칙을 통해 사용자 인터페이스를 결정하기 위한 정보를 제공하게 된다.

XML 문서를 정의할 때 속성과 원소의 구별은 문서를 정의하는 작성자의 선호도에 따라 이루어지게 된다[14].

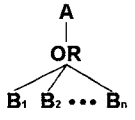

그림 4 자식 원소

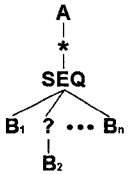

그림 5 반복과 옵션

즉, 동일한 내용을 어떤 작성자는 속성으로 표현하는 반면에 다른 작성자는 원소로 표현할 수 있다. 따라서, 속성과 #PCDATA 내용을 갖는 원소는 동일한 의미이며, 서로 교환되어 사용될 수 있다. DDG에서는 속성을 모두 원소로 변환해서 표현하는 방법을 사용한다. 정리 3-4는 속성을 원소로 변환하는 방법을 보여준다.

**정리 3.** 중간 노드가 속성을 갖는 경우

원소 A의 속성 $b_1$, $b_2$, ..., $b_n$은 #PCDATA 내용을 갖는 원소 A_$b_1$, A_$b_2$, ..., A_$b_n$로 변환되고, 생성된 원소들은 SEQ로 A의 자식 원소로 추가된다. 이때 속성이 #IMPLIED로 정의된 경우에 생성된 원소는 옵션('?') 특성을 갖는다. □

예를 들어, 그림 6과 같이 원소 A와 속성들이 정의되어 있다고 가정하자. 원소 A는 자식 노드들과 속성들을 포함하고 있다. 속성 $b_2$는 #IMPLIED로 정의되어 있다.

```
<!ELEMENT A  ( B1, B2, .. , Bn )*>
<!ATTLIST  A
           b1  CDATA   #REQUIRED
           b2  CDATA   #IMPLIED
           .....
           bn  CDATA   #REQUIRED>
```

그림 6 속성을 갖는 원소

그림 6의 원소와 속성은 정리 3에 의해 그림 7과 같은 형태로 변환된다. #IMPLIED로 정의된 속성 $b_2$에 해당되는 자식 노드 A_$b_2$는 옵션 특성을 가지고 있다.

```
<!ELEMENT A ((A_b1, A_b2?, .. , A_bn),
                B1, B2, .. , Bn )*>
<!ELEMENT A_b1  (#PCDATA)>
 ......
<!ELEMENT A_bn  (#PCDATA)>
```

그림 7 속성이 없는 원소

**정리 4.** 단말 노드의 속성

원소 내용으로 #PCDATA를 갖고, 속성들이 있는 경우에 원소는 자식 노드들을 갖는 원소로 변환된다. 또한 #PCDATA 내용은 새로운 원소로 정의되어 원래 원소의 자식 원소로 표현된다. □

그림 8의 원소 A는 원소 내용으로 #PCDATA를 갖고, 속성들을 포함한다.

원소 A는 정리 4에 의해 그림 9와 같이 변환될 수 있다. 그림 8의 원소 내용 #PCDATA는 $A_0$라는 자식 노드로 변환되었다.

```
<!ELEMENT A  ( #PCDATA )>
<!ATTLIST  A
           b1  CDATA   #REQUIRED
           b2  CDATA   #IMPLIED
           .....
           bn  CDATA   #REQUIRED>
```

그림 8 속성을 갖는 #PCDATA 원소

```
<!ELEMENT A (A0, A_b1, A_b2?,
                .. , A_bn)>
<!ELEMENT A0     (#PCDATA)>
<!ELEMENT A_b1   (#PCDATA)>
 ......
<!ELEMENT A_bn   (#PCDATA)>
```

그림 9 속성이 없는 원소

속성을 원소로 변환하는 것은 알고리즘 1을 통해서 이루어질 수 있다. 속성에 #IMPLIED가 사용된 경우에는 생성되는 원소에 옵션 특성을 기술하게 된다.

알고리즘 1. Attribute2Element

```
A : 현재 원소
B : 원소 A에 정의된 속성들의 집합
isRequired(b) : 속성 b가 #REQUIRED로 지정되어 있으면 true 리턴
childType(A) : 원소 A의 자식 노드 타입을 리턴
BEGIN
  switch(childType(A))
    case PCDATA:
      원소 A의 내용을 "<!ELEMENT A (A0)>" 형태로 변환한다.
      #PCDATA를 내용으로 갖는 새로운 원소 "A0"를 정의한다.
    case ANY:
      원소 A의 내용을 "<!ELEMENT A ( A0 )>" 형태로 변환한다.
      ANY를 내용으로 갖는 새로운 원소 "A0"를 정의한다.
  end-switch
  foreach  bi ∈ B
    #PCDATA를 내용으로 갖는 새로운 원소 "A_bi"를 정의한다.
    if ( isRequired(bi) == true )
      "A_bi" 이름을 A의 내용에 SEQ를 이용해서 추가한다.
    else
      "A_bi" 이름을 A의 내용에 옵션 특성과 함께 SEQ를 이용해서 추가한다.
    end-if
  end-foreach
END
```

정의 1과 정리 1-4를 이용하면 DDT를 DDG로 변환할 수 있다. 명함을 위한 XML 문서의 DTD를 예로 들어보자. 그림 10은 명함을 위한 DTD 예이다. 명함(namecard)은 순차(SEQ)를 통해서 이름(name), 회사

```
<!ELEMENT namecard   (name, company, contact)>
<!ELEMENT name       (#PCDATA)>
<!ELEMENT company    (name, address)>
<!ELEMENT address    (country?, city, street, zipcode)>
<!ELEMENT country    (#PCDATA)>
<!ELEMENT city       (#PCDATA)>
<!ELEMENT street     (#PCDATA)>
<!ELEMENT zipcode    (#PCDATA)>
<!ELEMENT contact    (phone | cellphone | email)+ >
<!ELEMENT phone      (#PCDATA)>
<!ELEMENT cellphone  (#PCDATA)>
<!ELEMENT email      (#PCDATA)>
```

그림 10 명함 DTD 예

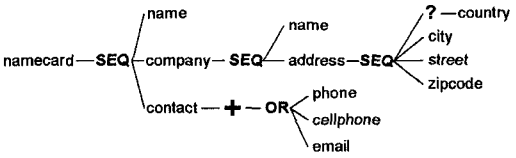


그림 11 명함 DTD의 DDG 표현

정보(company), 연락처(contact)로 구성되어 있다. 회사 주소(address)는 국가명(country)을 옵션으로 기술할 수 있다. 연락처는 전화(phone), 핸드폰(cellphone), 전자우편(email) 중에서 하나를 선택할 수 있으며, 여러 번 기술할 수도 있다.

그림 10의 명함을 위한 DTD는 그림 11과 같은 DDG로 표현할 수 있다. 그림 11에서 볼 수 있듯이 DDG에서 #PCDATA를 내용으로 갖는 원소들은 단말 노드로 표현되고, 자식 원소를 갖는 원소와 문서 구조 요소는 중간 노드들을 구성한다. 원소에 포함된 속성은 단말 노드로 표현된다.

DTD에서 한번 선언된 원소는 다른 여러 개 원소의 서브 원소로 사용될 수 있다. 이처럼 한 원소가 여러 곳에서 사용되면 DTD는 DDG로 표현하는 경우에 사이클이 있는 그래프 형태로 표현된다. 즉, 선언된 원소를 여러 곳에서 사용되는 경우에 사이클이 발생하게 된다. 따라서 DDG에서는 사이클이 발생하는 경우에는 사이클을 제거하기 위해서 반복적으로 사용되는 노드에 이전에 이미 처리되었다는 표시를 기술한다. 유사하게 XML의 DTD는 엔티티를 이용해서 재사용할 수 있다. 그림 12는 많은 사람들의 명함 정보를 포함하는 명함첩을 표현하기 위한 DTD이다. cardbook 원소는 명함첩의 소유자 정보와 명함 정보를 포함하고 있다. 명함 정보는 앞에서 정의한 namecard 원소를 재사용한다.

명함첩 DTD는 그림 13과 같은 DDG로 표현될 수 있다. 이미 정의되어 있는 원소들은 DDG에서 회색으로 채워진 사각형으로 표현한다.

```
<!ENTITY  % cardref  SYSTEM "namecard.dtd">
<!ELEMENT cardbook (owner, data)>
<!ELEMENT owner    (name, contact)>
<!ELEMENT data     (namecard)*>
%cardref;
```

그림 12 명함첩 DTD

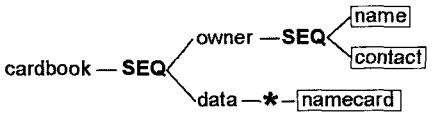


그림 13 명함첩의 DDG

XML 문서의 DTD는 DDG를 통해서 쉽게 그래프 형태로 표현될 수 있으며, 기존에 정의된 그래프를 재사용할 수 있는 방법을 제공한다.

### 3.2 프리젠테이션 모델

프리젠테이션 모델은 위젯(widget)들을 이용해서 사용자 인터페이스의 형태를 기술한다. 프리젠테이션 모델은 태스크 모델에 인터페이스 매핑 규칙을 적용함으로써 GUI 응용프로그램을 자동적으로 생성할 수 있다.

태스크 분석을 통해서 만들어진 DDG에서 단말 노드들은 실제 데이타를 입력받는 부분들이다. 단말 노드들은 사용자와 상호작용하면서 데이타를 입력받을 수 있는 GUI 컴포넌트로 표현될 수 있다. 이처럼 사용자와 상호작용하면서 단일 데이타를 입력받을 수 있는 GUI 컴포넌트들을 단순 컴포넌트(simple component)라고 한다. 예를 들어, 텍스트 필드, 체크박스, 콤보박스 등은 대표적인 단순 컴포넌트이다.

DDG의 중간 노드들은 단순 컴포넌트들을 그룹핑하기 위해서 사용되고, 그룹핑을 위해서 사용되는 GUI 컴포넌트들을 그룹 컴포넌트(group component)라고 한다. 그룹 컴포넌트의 예로는 패널, 탭팬 등이 있다. 그룹 컴포넌트는 내부에 여러 개의 단순 컴포넌트와 그룹 컴포넌트를 포함할 수 있다. 일부 중간 노드들은 자식 노드들과 함께 단순 컴포넌트로 표현될 수 있다. 이러한 단순 컴포넌트의 예로는 테이블이 있다.

DDG의 중간 노드들을 어떤 경우에는 창 단위로 GUI 컴포넌트들을 그룹핑한다. 이처럼 창은 PU(Presentation Unit)이라고 한다. PU는 여러 개의 단순 컴포넌트와 그룹 컴포넌트들을 포함하고 있다. 예를 들어, 창과 다이얼로그는 대표적인 PU들이다.

그림 14는 프리젠테이션 모델을 계층적으로 표현한 것이다. XML 문서는 여러 개의 PU들로 구성되어 있고, 각 PU는 다시 여러 개의 단순 컴포넌트와 그룹 컴포넌트들을 포함하고 있다.

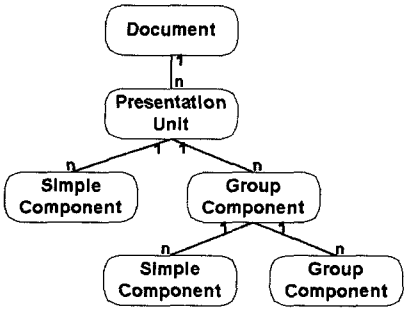


그림 14 프리젠테이션 계층적 구조

DTD의 요소들은 다양한 형태의 GUI 컴포넌트를 통해서 표현될 수 있다. 그림 15는 DTD 요소와 매핑될 수 있는 GUI 컴포넌트들 목록을 보여준다.

태스크 분석을 통해서 만들어진 DDG는 UI 디자이너와 상호작용을 통해서 DDDG(Decorated DDG)로 변환될 수 있다. DDDG는 DDG의 각 노드에 GUI 컴포넌트에 대한 정보를 추가한 것이다.

**정의 2.** DDDG

DDG에 GUI 컴포넌트에 대한 정보를 추가한 DDDG는 DDDG = <DDG, UI, PM>이다. UI는 XML 편집기에서 사용 가능한 GUI 컴포넌트들의 집합이다. PM는 DDG의 노드에 대한 GUI 컴포넌트의 매핑이다. PM = { ($<X_i, X_j, ..., X_n>, U_k$) | $X_i, X_j, ..., X_n \in V$, $U_k \in UI$ } 이다. □

그림 16은 명함 DDG에 GUI 컴포넌트에 대한 정보를 추가한 DDDG이다. SEQ 노드는 주로 이름과 테두리가 있는 그룹 컴포넌트로 묶이게 된다. 이 그룹 컴포넌트 내부에는 다른 그룹 컴포넌트나 단순 컴포넌트들이 존재할 수 있다. 단말 노드들은 직접 데이타를 입력할 수 있는 텍스트 필드를 갖는다. 가장 상위 노드는 윈도우 컴포넌트를 갖는다.

그림 17은 명함첩 DDG에 GUI 컴포넌트에 대한 정보를 추가한 DDDG이다. data 원소의 자식 노드(*, namecard)는 많은 데이타들을 반복적으로 입력해야 하기 때문에 데이타를 반복적으로 입력할 수 있는 그룹 컴포넌트를 사용한다.

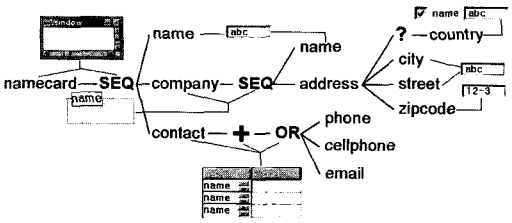


그림 16 명함의 DDDG

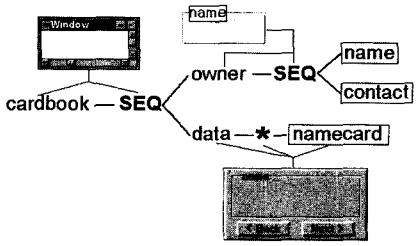


그림 17 명함첩의 DDDG

XED는 UI 디자이너로부터 정보를 받아 DDDG를 작성하게 된다. 그런데, UI 디자이너가 GUI 컴포넌트에 대한 정보를 주지 않는 경우에는 알고리즘 2를 이용해서 자동적으로 DTD에 적합한 GUI 컴포넌트를 결정하게 된다.

알고리즘 2. DetermineUI

child(x) : 노드 x의 자식 원소들의 집합
height(x) : 노드 x의 높이, 단말 노드는 y의 height(y)는 0이다.
is_root(x) : 노드 x가 루트 원소인지 여부
type(x) : 노드 x의 DTD 요소 타입
att_type(x) : 속성 노드 x의 데이타 타입
node : 현재 노드

| DTD 요소들 | GUI 컴포넌트 |
|---|---|
| #PCDATA | 텍스트 필드, 텍스트 영역, 콤보 박스, 트리 셀, 테이블 셀, 슬라이더, 스피너, 게이지 |
| 혼합된 내용 | 텍스트 영역 |
| ANY | 텍스트 영역 |
| 옵션 원소 | 체크 박스와 데이타 영역 |
| 자식 원소를 갖는 원소 | 트리, 테이블, 컨테이너(패널, 탭팬, 카드레이아웃 등) |
| 그룹 | 컨테이너(패널, 탭팬, 카드레이아웃 등) |
| ENTITY, ENTITIES 속성 | 파일 선택 버튼, 텍스트 필드 |
| 열거형 속성 | 텍스트 필드, 콤보 박스, 리스트 |
| ID, IDREF, IDREFS, NMTOKEN, NKTOKENS, CDATA, NOTATION 속성 | 텍스트 필드 |

그림 15 DTD 요소와 GUI 컴포넌트의 매핑

```
BEGIN
switch (type(node)) :
    case ELEMENT:
        if(height(e) == 0)
            UI_COMPONENT = TEXTFIELD
        else
            if(is_root(node))
                UI_COMPONENT = WINDOW
            else
                UI_COMPONENT = GROUP_PANEL
            end-if
        end-if
    case ATTRIBUTE:
        switch(attr_type(node))
            case ENUM:
                UI_COMPOENT = COMBOBOX
            default:
                UI_COMPONENT = TEXTFIELD
        end-switch
    case ? :
        ∀e, e ∈ child(node),
        if (height(e) == 0)
            UI_COMPONENT = CHECKBOX & TEXTFIELD
        else
            UICOMPONENT = CHECKBOX & GROUP_PANEL
        end-if
    case + :
    case * :
        if (child(node) == SEQ)
            ∀e, e ∈ child(child(node))
            if (height(e) == 0)
                UI_COMPONENT = TABLE or MULTIPANEL
             else
                UI_COMPONENT = MULTIPANEL
            end-if
        else if (child(node) == OR)
            ∀e, e ∈ child(child(node))
             if (height(e) == 0)
                UI_COMPONENT = TEXTAREA or TABLE
            else
                UI_COMPONENT = WINDOW
            end-if
        end-if
end-switch
END
```

DDG에서 각 노드에 대한 사용자 인터페이스가 결정되면, 쉽게 XML 편집기를 생성할 수 있다. XED는 알고리즘 3의 recursive-descent 파싱 방법을 통해서 XML 편집기를 자동적으로 생성하게 된다. XED는 DE에 정의된 원소와 속성에 대해 $de_i()$ 함수를 정의한다. 다음에 DDG의 가장 상위 원소에 해당되는 root() 함수를 호출한다. root() 함수는 recursive-descent 파싱 방식으로 서브 노드들을 위한 함수를 호출하면서 XML 편집기의 GUI를 형성하게 된다.

알고리즘 3. XML Editor 생성

```
function UI de_i ( de_i ) {
    UI ui = determinUI()
    foreach e_i ∈ child(de_i)
        ui_e_i = e_i()
        ui.add( ui_e_i )
    end-foreach
}
BEGIN
foreach de_i ∈ DE
    함수 de_i()를 정의한다.
end-foreach
DDG의 루트 노드에 해당되는 함수를 호출한다.
END
```

GUI 컴포넌트의 레이아웃은 동적인 위치 지정 방법[15]을 사용할 수 있다. 그러나 GUI 컴포넌트의 위치와 크기가 사용자가 원하는 형태가 아닌 경우에는 직접 조작 방식을 이용해서 사용자가 지정할 수 있다.

## 4. 시스템 구성

XED는 우선 XML 문서의 DTD를 입력으로 받고, 태스크 분석을 통해서 DDG를 생성하게 된다. 생성된 DDG는 프리젠테이션 규칙을 통해서 DDDG로 변환된다. XED는 DDDG를 통해서 드래프트 버전의 XML 편집기를 생성하게 된다. 생성된 편집기는 UI 디자이너에 의해서 커스터마이징 과정을 거치게 된다. UI 디자이너는 GUI 컴포넌트의 그래픽 속성들을 지정하거나 변경할 수 있다. XED 시스템은 그림 18과 같은 형태로 구성되어 있다.

그림 19는 XED 시스템이 실행되는 화면이다. 그림의 좌측에는 DTD의 DDG가 나타나고, UI 디자이너는 DDG에서 프리젠테이션을 위한 규칙들을 지정할 수 있다. 그림 우측에는 생성되는 XML 편집기의 형태가 나타난다. 생성된 편집기는 UI 디자이너가 직접 조작(direct manipulation) 방법을 통해서 편집기 형태를 변경할 수 있다. XED는 또한 UI 디자이너가 XML 편집기에서 필요한 툴팁과 문맥 도움말(context help)을 지

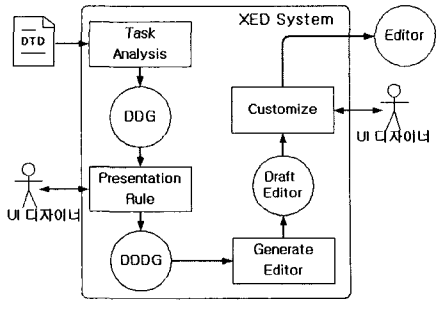


그림 18 XED 시스템 구성

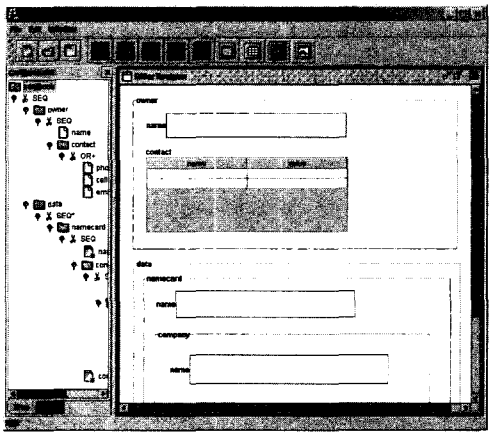

그림 19 XED 실행 화면

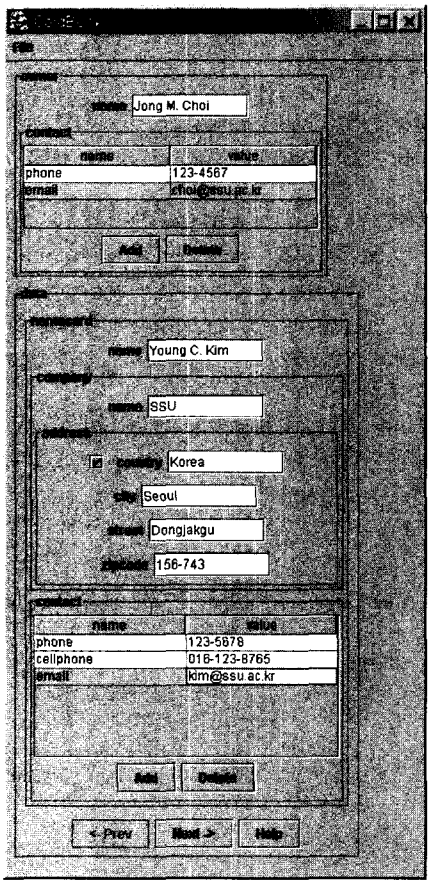


그림 20 CardBook 편집기

정할 수 있도록 지원한다. 툴팁과 문맥 도움말은 사용자가 XML 편집기를 올바르게 사용할 수 있도록 도움을 주게 된다.

| 내용 \ 편집기 | XMLSpy | SDE | CardBook |
|---|---|---|---|
| 사용의 편리성 | 40% | 4% | 56% |
| 배우기 쉬움 | 28% | 2% | 70% |

그림 21 설문 조사 결과

그림 20은 명함첩 DTD를 이용해서 자동 생성된 XML 편집기이다. CardBook에서 DTD의 각 필드들은 직접 값을 입력할 수 있는 GUI 컴포넌트들로 구성되어 있다.

폼 형태의 XML 편집기가 실제로 사용하기 편리한지 여부를 파악하기 위해서 25명의 XML 초보자들에게 XML 문서를 작성하고 설문지를 작성하는 경험적 테스트 방법[10]을 사용하였다. XMLSpy, XML SDE[3], CardBook 3개의 편집기를 이용한 사용자들의 설문 결과 XED를 통해서 생성된 CardBook 편집기가 가장 배우기 쉽고, 사용하기 쉬운 것으로 파악되었다. 그림 21은 설문 조사 결과이다.

또한 각 편집기를 이용해서 작성한 XML 파일들을 조사한 결과 XMLSpy와 SDE를 이용해서 작성한 문서에는 의미적인 오류들이 포함되어 있는 것을 확인할 수 있었다. 예를 들어, name이라는 원소가 같기 때문에 회사 이름을 기록해야 할 부분에 사람 이름을 기록한 오류를 여러 개 찾을 수 있었다.

## 5. 결 론

XML은 표준으로 제정된 이후에 컴퓨터뿐만 아니라 산업 전 분야에서 데이타를 표현하기 위해서 사용되고 있다. 따라서 이제 XML 문서는 컴퓨터 전공자가 아닌 일반인들도 작성해야할 필요성이 커지고 있다. 현재 널리 사용되는 XML 편집기는 범용 XML 문서를 작성할 수 있다는 장점을 가지고 있지만, 사용자가 XML 문법과 XML 문서의 의미를 알아야 올바르게 작성할 수 있다는 단점이 있다. 한편 XML 문서 중에서 상당 부분은 데이타 중심 문서이며, 이러한 문서인 경우에 폼 기반 GUI 편집기로 쉽게 XML 문서를 작성할 수 있다. 폼 기반 XML 편집기는 사용자가 문서의 내부적인 구조나 어휘를 모르면서 필요한 데이타만 입력하면서 XML 문서를 작성할 수 있는 장점을 가지고 있다.

논문에서는 데이타 중심 XML 문서의 DTD와 프리젠테이션 규칙이 주어지면, 모델 기반 인터페이스 개발 방법을 적용해서 자동적으로 폼 형태의 XML 편집기를 생성하는 방법과 XML 편집기 생성기를 소개하였다. XML 문서의 DTD 구조는 AND/OR 그래프를 변형한 DDG로 표현할 수 있으며, 태스크 모델을 통해서 DDG를 작성할 수 있다. 작성된 DDG는 프리젠테이션 모델을 통해서 DDG의 각 노드에 사용자 인터페이스 컴포넌

트에 대한 정보가 추가되면 자동적으로 XML 편집기를 생성할 수 있다. 생성된 XML 편집기는 폼 형태이기 때문에 사용자가 쉽게 데이타를 입력, 수정할 수 있도록 지원한다. UI 디자이너는 생성된 XML 편집기에 툴팁과 문맥 도움말을 제공함으로써 일반 사용자가 XML 편집기를 보다 쉽고, 올바르게 사용할 수 있도록 할 수 있다. 설문 조사 결과 폼 형태의 XML 편집기는 다른 편집기에 비해서 배우기 쉽고, 사용하기 쉽다는 결과가 나왔다.

현재 XED는 자바 기반으로 구현되었으며, 자바 스윙을 이용해서 XML 편집기를 생성하기 때문에 프로그래밍 언어에 종속적인 단점이 있다. W3C는 플랫폼과 프로그래밍 언어에 독립적인 XForms라는 웹 폼 사용자 인터페이스를 위한 표준을 작성하고 있다[16]. 향후에 XED에서 XForms를 지원하는 방법을 고려할 것이다.

## 참 고 문 헌


[1] XML Spy, available at http://www.xmlspy.com/

[2] Xeena, available at http://www.alphaworks.ibm.com/tech/xeena

[3] 신경희 and 유재우, "다중뷰를 지원하는 구조적 XML 에디터 생성", in *프로그래밍 언어 논문지*, 정보과학회, 2001.

[4] Joanna McGrenere, Ronald M. Baecker, and Kellogg S. Booth, "An Evaluation of a Multiple Interface Design Solution for Bloated Software," in *CHI'02*, pp.163-170. 2002.

[5] Norber Fuhr and Gerhard Weikum, "Classification and Intelligent Search on Information in XML," in *IEEE Data Engineering Bulletin*, pp. 51-58, 2002. available at http://www.is.informatik.uni-duisburg.de/publications/

[6] SUN, J2EE SDK, available at http://java.sun.com/j2ee/

[7] IBM, XML EditorMaker, available at http://www.alphaworks.ibm.com/tech/xmleditormaker

[8] Griffiths T., et al, "An Open Model-Based Interface Development System: The Teallach Approach," in *DSV-IS'98*, pp. 32-49, 1998.

[9] Peter J. Narclay, et. al., "The Teallach Tool: Using Models for Flexible User Interface Design," in *CADUI'99*, pp. 139-158, 1999.

[10] Ben Shneiderman, *Designing the User Interface -3rd ed*, Addison Wesley, 1998.

[11] Jean Vanderdonckt, "Using Data Flow Diagrams for Supporting Task Models," in *DSV-IS'98*, June, 1998.

[12] Francois Bodart, et. al., "Key Activities for a Development Methodology of Interactive Applications," in *Critical Issues in User Interface Systems Engineering*, Springer-Verlag, 1996.

[13] Michael F. Kleyn and Indranil Chakravarty, "EDGE - A Graph Based Tool for Specifying Interaction," in *UIST'89*, pp. 1-14, 1989.

[14] "SGML/XML: Using Elements and Attributes," available at http://www.oasis-open.org/cover/elementsAndAttrs.html

[15] Francois Bodart, et al, "Towards a Dynamic Strategy for Computer-Aided Visual Placement," in *CHI'94*, pp.78-87, 1994.

[16] "XForms - The Next Generation of Web Forms," available at http://www.w3.org/MarkUp/Forms/



최 종 명

정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 3 호 참조

유 재 우

정보과학회논문지 : 소프트웨어 및 응용
제 30 권 제 3 호 참조