UML 클래스 다이어그램을 XML DTD로의 변환 시스템 설계 및 구현

홍 도 석[†] · 하 얀[††] · 김 용 성[†††]

요 약

UML(Unified Modeling Language)의 구조 중에 UML 클래스 다이어그램은 객체모델링에 매우 적합하며, 최근에는 UXF(UML eXchange Format)까지 등장하여 UML 클래스 다이어그램을 여러 이기종 문서와의 교환이 가능하게 되었다. 따라서 본 논문은 UML 클래스 다이어그램을 인터넷 문서의 표준으로 자리잡은 XML 문서의 DTD 형태로 변환하는 시스템을 설계하였다. 이를 통하여 우리는 모델링 언어의 표준인 UML 클래스 다이어그램을 재사용 성이 뛰어난 XML 문서 형태로 쉽게 변형 및 저장할 수 있을 것이다. 또한 DTD로 변환하기 때문에 문서의 논리구조를 다양한 형식으로 표현할 수 있는 유연성을 제공할 수 있을 것이다.

Design and Implementation of Conversion System from UML Class Diagram to XML DTD

Do-Seok Hong[†] · Yan Ha[††] · Yong-Sung Kim[†††]

ABSTRACT

The UML(Unified Modeling Language) Class Diagram which is a part of structure of UML is fit for Object Modeling, and more recently, as the appearance of UXF(UML eXchange Format) UML Class Diagram by itself, can be exchanged in many other different system document. So this paper suggest the conversion system from UML Class Diagram to XML DTD. As this we can easily transformation and saving the UML Class Diagram that is the standard of Modeling Language to XML document which is so reusable. Also it can give a flexible method for the representation to the logical structure of document in various way because of converting XML DTD.

1. 서 론

고도화된 정보화 사회에서 정보 교환의 중요성이 심화되어 감에 따라 CALS/EC(Commerce At Light Speed/Electronic Commerce), 전자도서관(Digital Library), EDI(Electronic Data Interchange) 등의 분야에서와 같이 이 기종간의 상호 문서 교환을 목적으로 SGML(Standard Generalize Markup Language)이 탄생하게 되었다[1]. 그러나, SGML은 문법이 매우 복잡하며 그 용량이 방대하기 때문에 SGML의 독특한 특성을 그대로 반영한다는 것이 매우 어렵다. 이러한 단점을 보완하기 위하여 인터넷 표준으로 W3C에서 1998년에 공식적으로 제정한 XML이 등장하게 되었다[2].

XML은 SGML의 특성을 그대로 반영하는 반면 SGML에서 복잡하고 사용되지 않는 부분들을 제거하였으며, 인터넷에서 활용되어지는 HTML의 모든 기능과 속성들을 포함하고 있기 때문에, XML은 SGML과

† 정 회 원 : 전북대학교 대학원 전산통계학과
†† 준 회 원 : 중앙대학교 정보통신연구소 교수
††† 종신회원 : 전북대학교 컴퓨터과학과 교수
논문접수 : 2000년 6월 26일, 심사완료 : 2000년 11월 20일

HTML의 장점을 수용하고 한계를 극복한 새로운 인터넷 표준이라 할 수 있다[3].

한편, UML(Unified Modeling Language)은 객체지향 분석과 설계를 위한 대표적인 모델링 언어로서, Booch, Rumbaugh, Jacobson 등의 객체지향 방법론에 관한 석학들이 내어놓은 방법론을 통합한 것이다. 현재, UML은 객체 기술에 관한 국제 표준화 기구인 OMG (Object Management Group)에서 이미 산업계의 표준으로 인정하고 있다.

특히, UML의 구조 중에 UML 클래스 다이어그램은 객체들의 타입(클래스)들을 표현하며 그 클래스들의 정적인 관계를 표현하고 있기 때문에, 객체 모델링에 있어서 확고한 위치를 차지하고 있는 실정이다. 게다가 최근에 와서는 UXF(UML eXchange Format)의 등장으로 UML 클래스 다이어그램 자체를 이 기종간에 교환할 수 있게 되었다. 그만큼 UML 클래스 다이어그램의 활용 가치가 높아져서 여러 방면에 적용이 가능하게 되었다[4].

그러므로 본 논문에서는 UML 클래스 다이어그램을 XML DTD로 변환하는 시스템을 설계 및 구현하였다. UML 클래스 다이어그램을 DTD 형태로 변환하기 때문에 변환 및 저장이 용이한 XML 문서로의 확장이 용이할 것이다.

본 논문의 구성을 살펴보면 다음과 같다. 먼저 2장에서는 XML과 UML 클래스 다이어그램에 대하여 살펴보며 3장에서는 관련 연구로서 SGML과 XML 문서의 모델링 방법에 대하여 살펴본다. 4장에서는 UML 클래스 다이어그램을 DTD로 변환하는 시스템을 설계하기 위하여 변환 규칙과 알고리즘을 기술하였으며, 5장에서는 변환 시스템의 설계 및 구현된 시스템의 결과를 보여준다. 마지막으로 6장에서는 결론과 향후 연구과제에 관해서 논의한다.

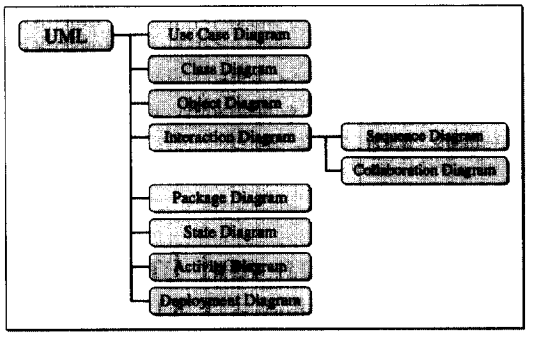
2. UML 클래스 다이어그램과 XML

이 장에서는 UML의 개요와 클래스 다이어그램의 구조, 그리고 XML의 구성 및 SGML과의 관계성 등에 대하여 살펴본다.

2.1 UML(Unified Modeling Language)

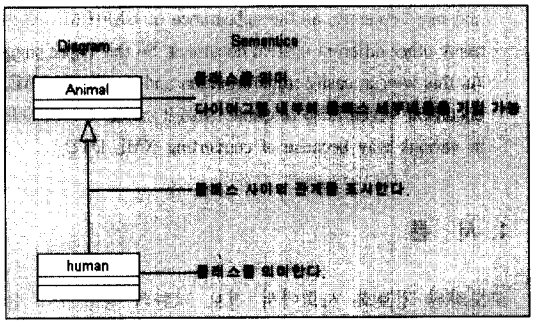
UML은 객체지향 시스템 모델을 작성하기 위한 객체지향적 분석과 설계를 개념 및 표기법 등을 제공하는 모델링 언어로서, Rumbaugh(OMT), Jacobson, Booch 등의 객체지향 방법론(methodology)에 관한 통합이다. 또한 객체 기술을 연구하는 국제 표준화 기구(OMG)에 의해 표준으로 인정받았다[9, 10].

UML의 전체적인 구성요소들은 (그림 1)과 같으며, 이러한 UML의 구성요소 중에서 특히 'Class Diagram'은 시스템의 정적인 설계의 사용하는 다이어그램으로 시스템 설계에 핵심적인 역할을 차지하고 있기 때문에 주로 다루어지고 있다.



(그림 1) UML의 구성요소

UML에서는 표기하려는 대상을 (그림 2)에서와 같이 다이어그램을 사용하여 나타내고 그 대상에 의미를 부여한다.



(그림 2) UML의 클래스 다이어그램과 의미

이와 같이, UML은 여러 가지 객체지향 개념을 결합한 형태로 다양한 메커니즘을 제공하고 있으며, 그 중에서 DTD를 모델링하기 위해서 제한조건(constraint)과 주석(comment), 엘리먼트 속성(element properties),

스테레오타입(stereotypes), 다형성(polymorphism)등의 메카니즘이 필요하다.

2.1.1 UML 클래스 다이어그램

UML 클래스 다이어그램에는 '사용 사례(Use case) 다이어그램', '정적 구조(Static Structure) 다이어그램', '순차(Sequence) 다이어그램', '협력(Collaboration) 다이어그램', '상태(State) 다이어그램' 등이 있다[9]. 이 중에서 가장 중심이 되는 것은 '정적 구조 다이어그램'으로 시스템을 논리적인 관점에서 분석하며, 클래스 다이어그램과 객체 다이어그램이 있다.

클래스 다이어그램은 정적 관계로 연결된 클래스들과 그 구성 요소들을 다이어그램으로 표현하며 객체 다이어그램은 클래스 다이어그램의 인스턴스(instance)이다.

UML 클래스 다이어그램은 여러 객체들의 타입, 즉 클래스를 표현하고 있으며 각각의 클래스들간의 정적인 구조를 나타낸다.

<표 1>은 UML 클래스 다이어그램을 구성하고 있는 주요 구성요소들에 대한 설명이다.

〈표 1〉 UML 클래스 다이어그램의 구성요소

구성요소		의 미
클래스		모델링 하고자 하는 시스템의 내부 개념을 표현하며, 클래스 이름, 에트리뷰트(attribute), 오퍼레이션(operation)으로 구성되어 있다.
관계성	집단화	클래스 사이를 연결하는 것으로 간단한 선으로 연결됨을 표시한다.
	일반화	하나 또는 다수 클래스의 구조 및 행위를 공유하는 클래스들간의 관계를 정의한다.
스테레오 타입		다른 취지로 사용할 하위 클래스로서 기본 클래스에 부가적인 제약 사항을 추가한 것이다.
다중성		상위 클래스를 구성하는 하위 클래스의 발생 횟수를 나타낸다.

2.1.2 Rational Rose

Rational Rose는 객체지향 개발 방법론을 적용하여 시스템을 개발할 때 소프트웨어 시스템을 분석하고 설계하는데 사용되는 도구이다. 기본적으로 모델/뷰 구조에 근거하여 소프트웨어 시스템을 설계하고 프로그램 코드를 생성하여 주며, 반복개발을 위한 제반 여건을 마련하고 있다.

2.2 XML(eXtensible Markup Language)

XML은 W3C에서 제정한 인터넷 표준으로, SGML의 특성을 그대로 반영하는 반면 복잡하고 사용되지 않는 부분들을 제거한 SGML의 부분집합(subset)이라 할 수 있다[2, 11]. 예를 들면, SGML의 내용 모델에서 포함(inclusion)과 제외(exclusion)를 지원하지 않으며, 연결자로서 '&'를 지원하지 않는다.

또한, 인터넷에서 활용되어지는 HTML의 모든 기능과 속성들을 포함하기 때문에 SGML과 HTML의 장점을 수용하고 한계를 극복한 새로운 인터넷 표준이라 할 수 있다.

XML은 XML 선언부(declaration), XML 문서 형 정의부(Document type Definition), XML 문서 인스턴스부(Document Instance)로 구성되어 있다[11].

XML 선언부는 XML 문서의 서두 부분에 작성하며 반드시 선언되어야 하는 XML의 버전과 인코딩(encoding) 방법, 그리고 DTD를 선택적으로 지정할 수 있는 RMD(Required Markup Declaration)를 정의한다.

XML DTD는 XML에서 가장 중요한 부분으로 엘리먼트(element), 에트리뷰트, 엔티티(entity) 등의 3가지 구성 요소로 나눌 수 있으며, (그림 3)은 편지에 대한 DTD이다.

```
<!DOCTYPE letter    [
<!ENTITY MARK       - o   "Knowledge Lab"  >
<!ELEMEMT letter    - -   title,body,date,signature,ps?>
<!ATTLIST letter    size  NUTOKEN #REQUIRED>
<!ELEMEMT title     - o   (#PCDATA)>
<!ELEMEMT body      - o   (head?,content,tail?)>
<!ELEMEMT head      - o   (#PCDATA)>
<!ELEMEMT contnet   - o   (#PCDATA | p*)>
<!ELEMEMT tail      - o   (#PCDATA)>
<!ELEMEMT date      - o   (#PCDATA)>
<!ELEMEMT sign      - o   (#PCDATA)>
<!ELEMEMT ps        - o   (#PCDATA)>    ]>
```

(그림 3) 편지에 대한 XML DTD

엘리먼트 선언은 HTML의 태그에 해당하는 부분으로 DTD에서 중심으로 문서의 구조를 나타내기 위한 공통 식별자를 선언하는 부분에 해당되기 때문에 문서의 논리적 구조를 파악할 수 있다. 애트리뷰트 선언은 문서나 엘리먼트의 속성을 정의하는 것으로 엘리먼트 상태, 문서의 텍스트 출력 형식 등의 속성을 선언한다. 엔티티 선언은 문서 내에서 참조될 수 있는 문자 집합의 단위로 일반 엔티티(general entity)와 매개변수 엔

티티(parameter entity)로 나눌 수 있다.

XML DI는 문서의 내용을 포함하는 부분으로 DTD에서 정의된 엘리먼트와 애트리뷰트 리스트를 이용하여 태깅한 실제 문서로 작성되어 있다.

3. 관련 연구

현재 XML에 대한 연구가 국내외적으로 활발히 진행되고 있으며, 대표적인 연구 분야로는 XML 문서의 모델링에 관한 연구이다. SGML/XML 문서를 모델링하는 연구로는 구조도, XOMT 다이어그램, UML 클래스 다이어그램을 이용하는 방법이 있다. [8]에서는 XML 문법에 맞는 사상 규칙과 알고리즘을 재정의 하였으며, 링크 구조에 대해서는 사상 규칙뿐만 아니라 형식 모델과 클래스 다이어그램 모델링 함수도 제안하였다.또한 최근에는 XML 스크립트가 등장하여 기존의 XSL을 이용하여 XML 문서를 생성하는 것보다 쉽고 빠르게 XML 문서의 생성 및 변형이 가능하게 되었다[9]. [11]에서는 본 논문과 유사한 방식으로 UML 클래스 다이어그램을 SGML DTD로 모델링하는 연구도 수행되었다.

이렇듯 여러 방면으로 XML에 관한 연구가 진행되어 가고 있으나, 현재 사용되고 있는 객체지향 데이터베이스나 프로그램 언어들의 종류가 너무 많기 때문에 이들 각각에 대한 스키마나 코드를 생성할 수 없다는 점이 문제로 대두되었다. 그러므로 XML 문서를 스키마 코드로 변환이 용이한 UML로 변환 및 검증하는 연구의 필요성이 증대되고 있는 실정이다.

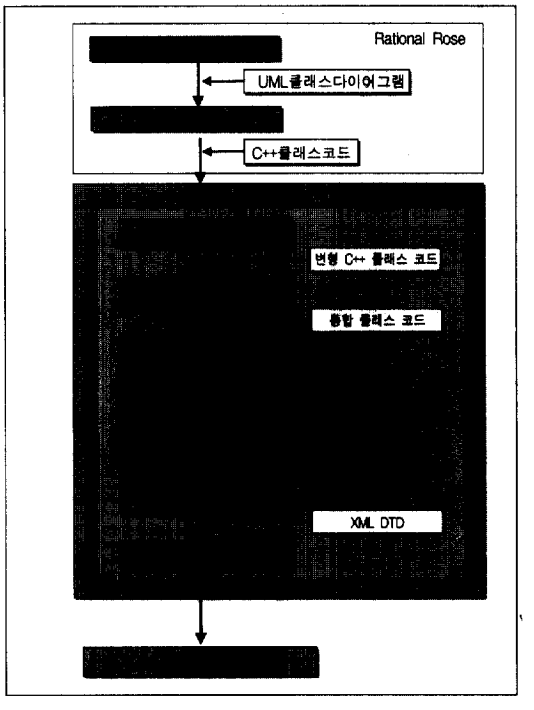
따라서, 본 논문에서는 UML 클래스 다이어그램을 XML DTD로 변환하는 시스템을 설계하고 구현하였다.

4. 시스템의 설계

이 장에서는 UML 클래스 다이어그램을 XML DTD로 변환하는 시스템을 설계하고 각각의 변환 규칙과 알고리즘에 대하여 기술한다. 먼저 시스템 구성을 살펴보면 (그림 4)와 같다.

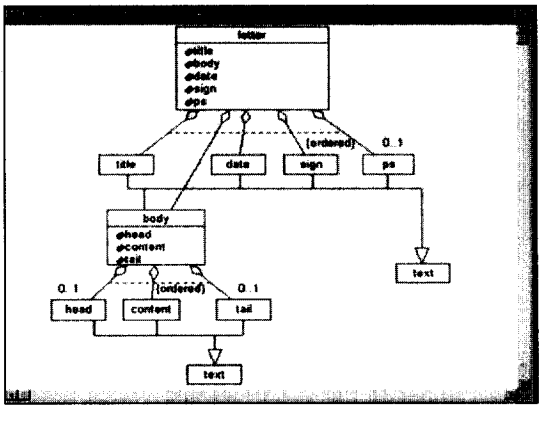
4.1 클래스 다이어그램 생성기

UML 모델링툴은 여러 가지가 존재하나 본 논문에서 입력으로 처리되는 UML 클래스 다이어그램은 현재 가장 많이 사용하고 강력한 기능을 보유하고 있는 Rational Rose2000을 사용하였다.



(그림 4) 시스템 구성도

참고로 (그림 5)는 본 논문에서 사용하고 있는 편지에 대한 UML 클래스 다이어그램을 Rational Rose2000을 이용하여 작성한 것이다.



(그림 5) 편지에 대한 UML 클래스 다이어그램

4.2 클래스 발생기

클래스 발생기는 입력한 UML 클래스 다이어그램을 순수한 C++ 클래스 형태로 바꾸는 작업을 수행하는 부분이다. 현재 UML 클래스 다이어그램을 다른 프로

그램 언어(C++, Java, Delphi 등)코드나 스키마 코드로 변환하는 프로그램은 많이 있지만 실제로 UML 클래스 다이어그램이 의미하고 있는 내용을 정확하게 파악하여 변환하지 못하고 있는 실정이다. 그러나 Rose의 C++코드 변환기를 통하여 생성된 UML 클래스 다이어그램에 대한 코드는 클래스를 포함하여 클래스의 오퍼레이션과 에트리뷰트까지 반영하고 있기 때문에 본 논문에서는 Rose의 C++ 코드 변환기를 사용하여 C++ 클래스 코드를 생성하였다.

4.3 XML DTD 변환기

XML DTD 변환기는 Rose의 C++ 코드 생성기를 통하여 생성된 UML 클래스 다이어그램에 대한 C++ 클래스 코드를 DTD로 변환하는 부분이다.

(그림 4)와 같이 XML DTD 변환기는 전처리기, 헤더파일 처리기, DTD 변환기, DTD 조작기로 구성되어 있으며, 각각의 기능을 살펴보면 다음과 같다.

4.3.1 전처리기

Rose의 C++ 코드 생성기를 이용하여 생성된 순수한 C++ 코드는 UML 클래스 다이어그램에 있어서 각각의 클래스들간의 계층성(부모와 자식관계)은 표현하고 있지만 동일한 레벨의 클래스들간의 순서와, 그러한 클래스들간의 관계성에 대해서는 표현하고 있지 않다. 또한 UML 클래스 다이어그램에서 자주 쓰이는 발생지시자에 대한 사항도 처리되고 있지 않다. 그러므로 전처리기에서 동일레벨 클래스들간의 관계성과 발생지시자를 처리하기 위해서 순수 C++ 코드에 다음과 같은 사항을 첨가하여 변형 C++ 코드를 생성한다.

(1) 동일레벨 클래스간의 관계성 처리

동일 레벨의 UML 클래스들간의 관계는 <표 2>에서 보이는 것처럼 '순서가 있는 관계', 'or 관계', '순서가 없는 관계'로 구성되어 있으며 실제로 XML DTD에서 "," , "|", "&"를 이용하여 표기하고 있다.

〈표 2〉 클래스간의 관계

연결자 타입	,	|	&
의미	순서가 있는 관계	or 관계	순서가 없는 관계

그러므로 전처리기에서 관계성을 처리하기 위해 UML 클래스들간에 해당되는 관계성에 관한 함수를 생성된 순수 C++ 코드에 첨가한다. 첨가되는 관계성 함수는 다음과 같은 구조로 정의한다.

```
connect_type("연결자 타입", 클래스 이름,…,클래스 이름)
```

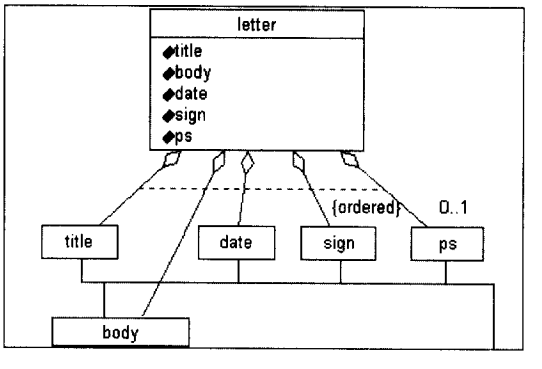
<표 3>은 (그림 6)과 같이 관계성이 포함된 UML 클래스 다이어그램의 C++ 코드에 관계성 함수를 첨가한 예이다.

〈표 3〉 letter.h에 관계성 함수를 첨가

```
class letter
        void set_title (void value);
                :
                :
        const void get_ps () const;
        void set_ps (void value);

        connect_type(",", title, body, date, sign, ps);
        // 관계성 함수를 첨가
  private :

        void title;
        void body;
        void date;
        void sign;
        void ps
                :
};
```



(그림 6) 관계성이 포함된 클래스 다이어그램

(2) 클래스의 발생지시자 처리

어떤 클래스에 대한 발생 지시자의 경우는 <표 4>에서 보이는 것처럼 "0번 이상(*)", "1번 이상(+)", "0번 혹은 1번(?)"으로 구성되어 있으며 실제로 XML DTD

에서 "*" , "+", "?"를 이용하여 표기하고 있다.

〈표 4〉 클래스의 발생지지사 타입

발생지시자 타입	?	*	+
의미	0번 혹은 1번	0번 이상 반복	1번 이상 반복

전처리기에서 발생지시자를 처리하기 위해 발생지시자를 포함하고 있는 클래스에 해당되는 발생지시자 함수를 생성된 순수 C++ 코드에 첨가한다. 첨가되는 발생지시자 함수는 다음과 같은 구조로 정의한다.

```
indicate_type("발생 지시자 타입", 클래스 이름)
```

<표 5>는 (그림 7)과 같이 발생지시자가 포함된 UML 클래스 다이어그램의 C++ 코드에 발생지시자 함수를 첨가한 예이다.

〈표 5〉 발생지시자가 첨가된 코드

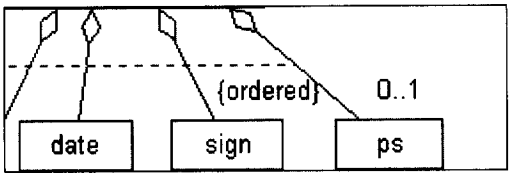
```
#include "letter.h"
#include "text.h"

class ps :  public text
{
  public:

       ps();
       ps(const Sign &right);
       ~ps();
          :
       ps & operator=(const ps &right);
       int operator==(const ps &right) const;
       int operator!=(const ps &right) const;

       indicate_type("?", ps) ; //발생지시자 함수 첨가
          :
};

inline const letter * ps::get_the_letter () const
{
  return the_letter;
}
 :
```



(그림 7) 발생지시자가 포함된 클래스 다이어그램

4.3.2 헤더파일 처리기

Rose C++ 변환기를 이용하면 cpp 파일과 헤더(header)파일 형태로 UML 클래스 다이어그램에 대한 코드를 생성한다. 여기서 cpp 파일은 클래스에 대한 생성자 및 소멸자에 관한 처리만 있기 때문에 XML DTD로 변형 시 거의 영향을 주지 않기 때문에 헤더파일만을 처리한다. 또한 변환작업의 용이하게 하기 위해서 각각의 헤더파일에서 클래스의 정보와 속성을 추출하여 통합된 하나의 클래스 코드를 생성한다. 헤더파일에서 클래스 정보를 추출하여 통합된 클래스 코드를 생성하는 절차는 다음과 같다.

① 최상위 부모 클래스로부터 하위 클래스까지 관계성에 기준 하여 순서를 결정한 후에 이에 해당되는 헤더파일을 정렬한다.
② 정렬된 각각의 헤더파일에서 생성자, 소멸자, 할당 연산자, 등치 연산자에 해당되는 내용을 삭제한다.
③ 정렬된 각각의 헤더파일에서 클래스 선언 부분과 클래스 에트리뷰트에 해당되는 데이터 멤버를 계층성 함수로 다시 표현하여 추출하고 전처리과정을 통하여 생성된, 관계성 함수와, 발생지시자 함수를 추출한다.
④ 추출된 내용을 통합하여 통합 클래스 코드를 생성한다.

위에서 설명한 클래스 코드 생성절차 알고리즘은 <표 6>과 같다.

〈표 6〉 통합 클래스 Code 생성 알고리즘

```
입력 : Header 파일
출력 : 통합 클래스 코드
begin {
 while(!empty_head_file()) {
        check_relation(parent, child);/*클래스간의
                          포함관계 처리 함수*/
        sort(head_file);/*헤더 파일 정렬*/
          while(!empty_head_file()) {
             delete(garbage_operator) ; /*헤더파일에서
                          불필요한 내용 제거*/
          classify(declaration_part, attribute_part);
       /*클래스 선언부분과 에트리뷰트처리 함수*/
           represent_relation();/*계층성 변환 함수*/
           import(relation_fun, indicate_fun) ; /*관계성
              함수와 발생지시자 함수 추출*/
           }
          while(!empty_modified_file()) {
           integrate(head_file);/*헤더파일 통합 */
           }
        }
} end
```

다음 <표 7>은 (그림 5)에 해당되는 변형 클래스 코드 'letter.h'를 헤더파일 처리기를 통하여 생성된 통합 클래스 코드를 보여주고 있다.

〈표 7〉 헤더파일 처리기를 통한 변환코드

class letter
{
public:
letter();
letter(const letter &right);
~letter();
 	
const void get_title () const;
void set_title (void value);
:
:
const void get_ps () const;
void set_ps (void value);
 	
connect_type(",", title, body, date, sign, ps);
 	
private :
 	
void title;
void body;
void date;
void sign;
void ps
);
	변환전코드

class letter { **private** : Child _Set(title, body, date, sign, ps) ; connect_type(",", title , body, date, sign) ; indicate_type(ps,:" ? ") ; };	변환후코드

4.3.3 DTD 변환기

DTD 변환기는 헤드파일 처리기를 통하여 생성된 통합 클래스 코드를 XML DTD로 변환하는 작업을 수행한다. 한편 통합 클래스 코드로부터 XML DTD를 생성할 때 다음과 같은 제한 조건을 두었다.

[제한조건1] 모든 클래스는 반드시 'ELEMENT'로만 변환을 시킨다. 즉 Entity로의 변환은 생각하지 않는다.

[제한조건2] '()'는 관한 처리는 생각하지 않는다.

[제한조건3] 'text'를 상속하는 클래스는 'ELEMENT'의 내용을 '(#PCDATA)'로만 간주한다.

따라서 DTD 변환기는 (그림 4)에서 제시되었던 것처럼 XML DTD 헤드 생성 단계, ELEMENT 헤드 생성 단계, 연결자 및 발생지지사 처리 단계로 구성되어 있다. 각각의 단계에 대하여 설명하면 다음과 같다.

(1) XML DTD 헤드 생성 단계

XML DTD 헤드 생성 단계는 XML의 버전 정보와 인코딩 정보를 포함하고 있는 XML DTD의 헤드를 생성하는 부분이다. XML DTD의 헤드는 <표 7>과 같다.

〈표 7〉 XML DTD의 헤드

<?xml version="1.0" encoding="EUC-KR"?>

(2) ELEMENT 헤드 변환 단계

ELEMENT 헤드 변환 단계는 통합 클래스에서 'class'로 표기되어 있는 코드를 XML DTD의 <ELEMENT> 태그로 변형하는 부분이다. XML DTD의 <ELEMENT> 태그의 기본적인 문법 구조를 기초로 하여 <ELEMENT> 테그를 생성한다. 변환 함수는 <표 8>로 정의한다.

〈표 8〉 Element 변환 함수

element_head(MDO, ele_tag, ele_name, o_tag) ;	
정의	element_head 함수는 MDO 생성과 ELEMENT 테그 및 ELEMENT 이름, 생략 테그를 생성한다.

(3) 연결자 처리 단계

연결자 처리 단계는 통합 클래스 코드에서 생성된 클래스들간의 관계성을 처리해 주는 단계로 동일 레벨의 클래스간의 관계성을 주로 처리한다. 동일 레벨의 클래스간의 관계성을 처리하기 위해서는 해당 클래스가 'Child_set'이라는 함수의 존재 여부를 살펴본 다음 'Child_set'이 있는 경우에만 동일 클래스들간의 관계성 함수를 처리하게 된다.

(4) 발생 지시자 처리 단계

발생 지시자 처리 단계는 각각의 클래스에 대한 발생지시자를 처리하는 단계로 연결자 처리 단계와 유사한 형태로 'child_indicate()' 함수의 여부에 따라 발생지시자에 관한 사항을 처리한다. 위와 같은 절차로 통합 클래스 코드안에 들어있는 모든 class에 대하여 위

의 절차를 반복하여 수행하여 XML DTD를 생성하게 된다.

<표 9>는 위에서 제시한 각각의 단계에 대한 알고리즘을 통합하여 정리한 표이다. XML DTD 헤드 생성기, ELEMENT 헤드 변환단계 및 연결자와 발생 지시자를 처리하는 단계로 구성되어 있다.

〈표 9〉 변환 알고리즘

```
입력 : 통합 클래스 코드
출력 : XML DTD
==========================================
 begin {
 make_XML_HEAD();
 doc_tag_flag=0;
 { while (TYPE_ID =="class")
   { if (!doc_tag_flag)
     { doctype(MDO, doctype_tag, doc_name, DSO);
       doc_tag_flag=1 ; }
     element_head(MDO, ele_tag, ele_name, o_tag) ;
   while(!class_content_empty()) /*연결자 처리
                                  모듈 시작*/
     { if(child_set!=NULL)
       { switch(connect_type)
         {
          case : ","  // ordered type
          make_connect_type(child_set, ",") ;
          case : "|"  // or type
          make_connect_type(child_set, "|") ;
          case : "&"  // not ordered type
          make_connect_type(child_set, "&") ;
          }
        if(child_indicate()) /*// 발생 지시자
                      처리 모듈 시작 */
          { switch(indicate_type) {
              case : "?"
                make_indicate_type(child, "?");
              case : "+"
                make_indicate_type(child, "+");
              case : "*"
                make_indicate_type(child, "*");
             } }
      element_body( child_set, indicate, MDC) ; }
    else { if(content_type!=NULL))
          element_body( content_type, MDC) ;
        } } }
   doctype_tail(DSC, MDC);
 } end
==========================================
```

최종적으로 XML DTD 변환기 알고리즘인 <표 9>를 통하여 생성된 편지의 UML 클래스에 대한 XML DTD는 <표 10>과 같다.

〈표 10〉 생성된 XML DTD

```
<?xml version="1.0" encoding="EUC-KR"?>
<!DOCTYPE letter [
<!ELEMEMT letter  - o    title,body,date,sign,ps?>
<!ELEMEMT title   - o    (#PCDATA)>
<!ELEMEMT body    - o    (head?,content,tail?)>
<!ELEMEMT head    - o    (#PCDATA)>
<!ELEMEMT contnet       - o (#PCDATA)>
<!ELEMEMT p       - o    (#PCDATA)>
<!ELEMEMT tail    - o    (#PCDATA)>
<!ELEMEMT date    - o    (#PCDATA)>
<!ELEMEMT sign    - o    (#PCDATA)>
<!ELEMEMT ps      - o    (#PCDATA)> ]>
```

<표 11>은 앞에서 언급되었던 모든 과정에 나오는 주요한 함수를 정리한 표이다.

〈표 11〉 변환 함수

함수위치	함수명	의 미
전처리기	connect_type()	동일 클래스간의 관계성 첨가함수
	indicate_type()	클래스의 발생지시자 처리함수
헤드파일 처리기	child_set()	클래스간의 계층성 처리 함수
DTD 변환기	make_ xml_head()	XML DTD의 헤드 생성함수
	doctype()	XML Document 헤드 생성함수
	element_head()	XML의 Element 헤드 생성함수
	make_connect_type()	연결자 생성함수
	make_indicate_type()	발생지시자 생성함수
	class_content()	클래스의 내용 식별 유무 판별 함수
	child_indicate()	자식클래스의 발생지시자 식별 함수
	element_body()	XML의 Element Body 생성함수
	doctype_tail()	XML Document Tail 생성함수

4.3.4 DTD 조작기

DTD 변환기를 통하여 생성된 XML DTD는 단순히 통합 클래스 코드에 기준 하여 이루어 졌기 때문에 정형화된 형태를 가질 수 없다. 따라서 DTD 조작기에서 생성된 XML DTD를 정형화된 형태로 변형시킨다. 예를 들면 XML DTD 변환기를 통해서 생성된 편지에 대한 XML DTD 가운데 <표 12>와 같은 비정형 XML

DTD 부분을 정형 XML DTD 형태로 간략화 시킨다.

〈표 12〉 DTD 정형화 예

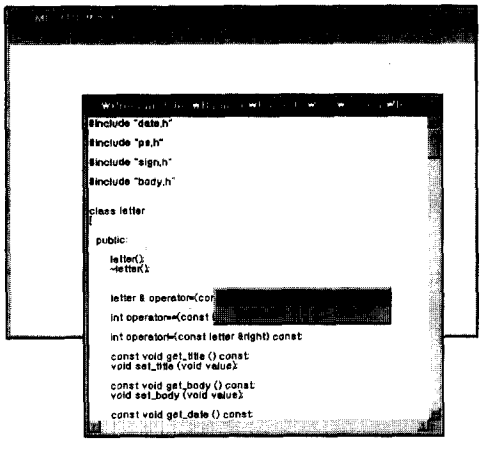
비정형 XML DTD	<!ELEMENT date -o (#PCDATA) > <!ELEMENT sign -o (#PCDATA) > <!ELEMENT ps -o (#PCDATA) >
정형 DTD	<!ELEMENT (date|sign|ps) -o (#PCDATA) >

5. XML DTD 변환 시스템의 구현

5.1 시스템 구현환경 및 실행 화면

본 논문에서 구현한 변환 시스템은 Window98 환경에서 Delphi 5.0을 사용하였다. 입력으로 사용되는 UML 클래스 다이어그램을 작성하기 위해서는 Rational社의 Rose2000 Enterprise 버전을 이용하였다. 각각의 실행 화면을 살펴보면 다음과 같다.

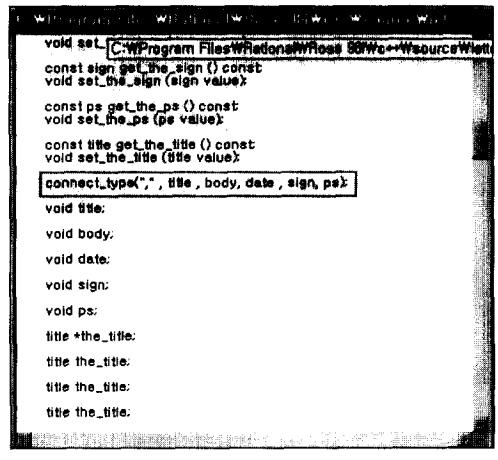
먼저 입력으로 사용하는 UML 클래스 다이어그램은 (그림 5)인 편지에 대한 UML 클래스 다이어그램이다. (그림 8)은 'letter.h' 파일을 입력으로 받아들여 동일 레벨의 UML 클래스들간의 관계성과, 각 클래스들이 가질 수 있는 발생지시자에 대하여 처리하고 있는 화면이다.



(그림 8) 관계성 처리 화면

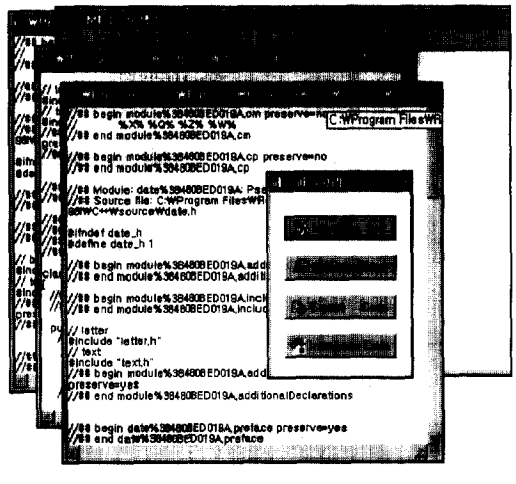
동일한 레벨에 있는 UML 클래스들간의 순서를 나타내기 위하여 관계성 함수를 추가시킨다. 그러면, (그림 9)와 같이 변형된 'letter.ch' 파일을 생성하게 되며 관계성 함수가 'letter.h'파일에 첨가되어 있는 것을 볼 수 있다.

헤더파일 처리기에서는 변형된 C++ 코드의 클래스 정보와 속성을 추출하여 통합된 하나의 클래스 코드를 생성한다.



(그림 9) 관계성이 첨가된 헤더파일

(그림 10)은 헤더 파일 처리기에서 수행되는 일련의 과정을 처리하기 위한 화면이다.

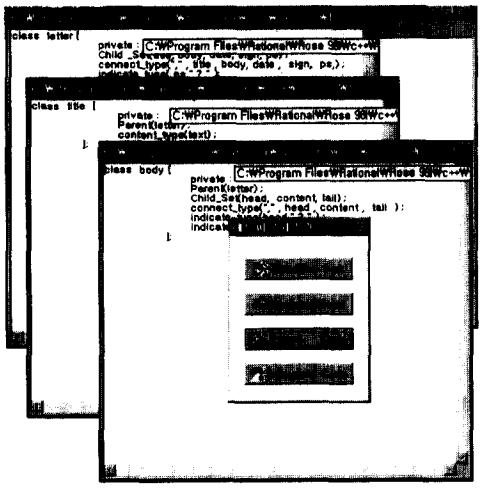


(그림 10) 헤더파일 처리기 화면

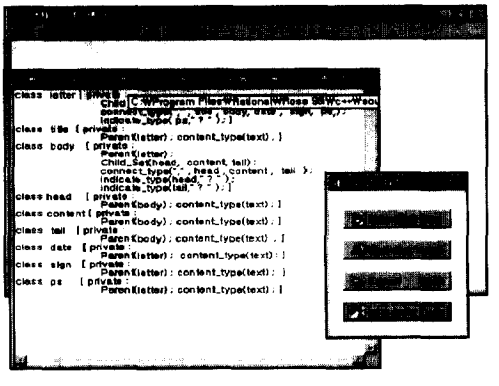
헤더 파일 처리기 과정을 통하게 되면 변형된 헤드 파일들이 생성이 되며, (그림 11)은 헤더파일 처리기결과를 보여주고 있는 화면이다.

XML DTD 변환기에서 헤더파일 처리기를 통하여 생성된 각각의 UML 클래스에 대한 코드를 합하면 통합 클래스 코드를 얻을 수 있으며, (그림 12)는 통합 클래스 코드를 생성하는 화면을 보여주고 있다.

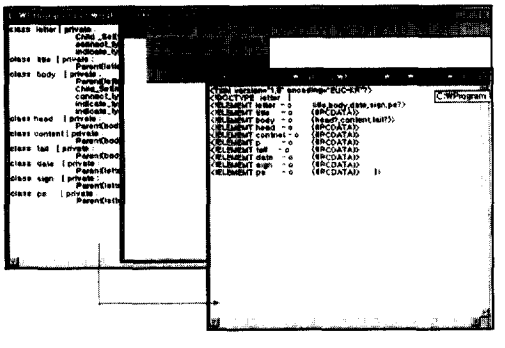
통합 클래스에 기준하여 생성된 XML DTD를 DTD 조작기를 거치게 되면, (그림 13)과 같이 정형화된 XML DTD가 생성된다.



(그림 11) 변형된 헤더파일



(그림 12) 통합 클래스코드 화면



(그림 13) XML DTD 생성화면

6. 결론 및 향후 연구과제

현재 UML은 객체지향 시스템 개발을 위한 표준적인 방법론으로 OMG의 승인을 받고 있으며, UXF의 등장으로 UML 클래스 다이어그램을 확장하여 많은 분야에 적용할 수 있게 되었다.

한편, XML은 이기종 시스템간의 문서 교환을 위해 탄생하였으며, SGML의 단점을 극복하고 확장성과 문서의 구조적인 표현이 불가능한 HTML의 단점을 보안한 새로운 인터넷의 표준으로 자리잡아 가고 있다. 더욱이 XML은 어떠한 형태의 문서라도 다양한 문헌의 모델링이 가능하며, 차세대 하이퍼텍스트 기능 및 문서의 내용과 스타일 정보를 분리하여 사용하는 특성을 가지고 있다. 또한 문서의 재사용성이 뛰어 나기 때문에 다방면에서 적용될 수 있는 언어인 것이다.

이렇듯 XML 문서에 대한 중요성이 가중되는 시점에서, 본 논문에서는 UML의 핵심이 되고 있는 UML 클래스 다이어그램을 XML DTD로 변환하는 시스템을 설계하였다. 즉, UML 클래스 다이어그램이 가지고 있는 객체지향 개념을 객체 모델과 유사한 형태를 가지고 있는 XML DTD 형태로 변환하였다.

XML DTD로의 변환 알고리즘을 통하여, UML 클래스 다이어그램에 대한 적용 범위를 확장시킬 수 있으며, UML 클래스 다이어그램을 재사용 성이 뛰어난 텍스트 형태의 문서로 저장할 수 있을 것이다.

향후 연구 과제로는 UXF를 통하여 확장된 수많은 UML 클래스 다이어그램을 XML DTD의 형태로 변환하거나, 변환된 DTD를 객체지향 데이터베이스에 저장하기 위한 일반적인 사상 알고리즘을 개발하는 것이다. 더 나아가서 UXF 구조의 브라우저나, JAVA, C++ 등의 소스 코드로부터 UXF를 생성할 수 있는 시스템을 개발하는 것이다.

참 고 문 헌

[1] 정회경, "SGML/XML의 개요", http://www. ccpack. or.kr/, 1998.

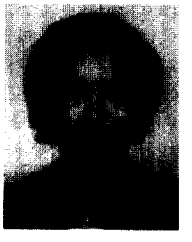
[2] W3C, "Extensible Markup Language(XML) 1.0," http://www.w3.org/TR/1998/REC-xml-1998210/, 1998.

[3] Jon Bosak, "XML, Java, and the futere of the Web," http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm/.

[4] 박종훈, "XML 응용과 제품개발 현황", 경영과 컴퓨터 1998. 9.
[5] 채원석, 하얀, 김용성, "UML 클래스 다이어그램을 이용한 XML 문서 구조 다이어그래밍", 한국정보처리학회 논문지, 제6권 제10호, pp.2670-2679, 1999. 10.
[6] G. Booch, "UML for XML Schema Mapping Specification," Dec 8, 1999.
[7] http://www.xmlscript.org/.
[8] Richard Lander, "ML : The New Markup Wave," http://www.csclub.uwa.terloo.ca.u/relander/XML/Wave/xml_mw.html/.
[9] E. Herwijnen, "Practical SGML," 2nd Edition, Kluwer Academic Publishers, 1994.
[10] 박인호 외 6, "XOMT : SGML DTD 설계를 위한 객체 다이어그래밍 기법", 정보과학회논문지(C), 제3권 제3호, pp.228-237, 1997. 6.
[11] Y. Ha, Y.-J. Hwang, Y.-S. Kim, "SGML DTD Modelling Using UML Class Diagram," Proceedings of the ICT'99, Vol.1, pp.206-210, June 1999.
[12] Rational Software Corporation, "UML Nonation Guide version 1.1," http://www.rational.com./, 1997. 9.
[13] 이원석, "XML Overview," KRIC, 1998.
[14] 주종철, "SGML/XML의 현재와 미래", http://www.ccpack.or.kr/, 1998.
[15] "HTML, SGML, PDF, XML : What is the differences ?," http://iai.sgml.com/ 971206-01.asp.
[16] Craig Larman, "Applying UML and Patterns," Prentice-Hall, 1998.
[17] Jean-Michel Bruel "Transforming UML Models to Formal Specifications," 1998.
[18] Paul Harmon, Mark Watson, "Understanding UML," 1997.
[19] Rational Software, "UML Semantics version 1.1," http://www.rational.com/uml/, 1997. 9.
[20] Rohit Khare and Adam Rifkin, "X Marks the Spot," http://www.cs.caltech.edu/~adam/papers/xml/x-marks-the-spot.html/.

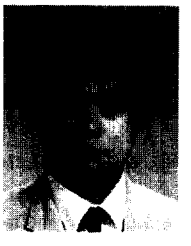
홍 도 석

e-mail : dshong@cs.chonbuk.ac.kr
1998년 전북대학교 컴퓨터과학과 졸업(이학사)
2000년 전북대학교 전산통계학과 졸업(이학석사)
현재 전북대학교 전산통계학과 박사 과정
관심분야 : UML, XML 모델링, KMS, 정보검색



하 얀

e-mail : hayan@object.cse.cau.ac.kr
1992년 덕성여자대학교 전산학과 졸업(이학사)
1994년 이화여자대학교 교육대학원 전자계산전공(교육학석사)
2000년 전북대학교 대학원 전산통계학과 졸업(이학박사)
2000년~현재 중앙대학교 정보통신연구소 연구전담교수
관심분야 : XML 응용, 객체지향 모델링, 컴포넌트 모델링, 애니메이션 컴포넌트, 전자도서관, 퍼지 정보검색



김 용 성

e-mail : yskim@moak.chonbuk.ac.kr
1978년 고려대학교 수학과 졸업(이학사)
1984년 광운대학교 전산학과(이학석사)
1992년 광운대학교 전산학과(이학박사)
1985년~현재 전북대학교 컴퓨터과학과 교수
1996년~1998년 한국학술진흥재단 전문위원
관심분야 : 디지털 도서관, 정보검색, 인터넷 기반 정보검색, 멀티미디어 시스템, 인공지능 등