OSD, CDF 문서로부터 UML 클래스 다이어그램으로 변환 시스템

하 얀†

요 약

최근 웹-기반 XML 응용에 관한 필요성이 급증함에 따라, 이와 관련된 WIDL, OSD, CDF에 관한 연구가 널리 이루어지고 있다. 특히, OSD 포맷은 소프트웨어 패키지와 그들 간의 관계를 정의하는 것으로 CDF 문서 등에서 많이 사용되고 있다. 그러나 아직까지 이에 대한 객체 모델링이 제안된 바 없어, 소프트웨어를 재사용하려는 사람들마다 별도의 양식으로 이들 관계를 표현하고 있는 실정이다. 따라서, 본 논문은 UML의 다양한 관련성을 이용하여 OSD와 CDF 문서를 객체 모델링하고자 한다. 이 연구는 XML 각 응용을 객체 모델링시키고 데이터베이스화하는 연구의 일환으로 OSD와 CDF 문서를 UML 클래스 다이어그램으로 사상시키고자 한다. 이를 통해 소프트웨어 패키지간의 구현과 실체 관계를 UML 형태로 일관되게 표현할 수 있을 뿐만 아니라 모델링 함수를 통해 XML 문서의 구조를 쉽게 파악, 변형을 용이하게 하는 장점이 있다.

Converting System from OSD and CDF documents to UML Class Diagram

Ha Yan†

ABSTRACT

For increasing needs about web-based XML applications, there has been a lot of studies about WIDL, CDF and OSD that are related to XML. Especially, the OSD format describes software packages and their inter-dependencies that is widely used in a CDF file. It uses different forms for representing relationships among software packages, because there has not been proposed a method of object modeling about it until now. In this paper, we propose a modeling method for the OSD and CDF documents using UML relationships. It is a part of the project to merge XML applications and convert the OSD and CDF documents into an UML class diagram to generate OODB scheme. We can easily not only visualize implementation and dependency relationships among software packages but also understand and transform logical structures of XML documents using modeling functions

키워드 : OSD, CDF, UML 클래스 다이어그램(UML Class Diagram), XMI, 객체 모델링(Object Modeling)

1. 서 론

XML은 엘리먼트, 애트리뷰트, 엔티티를 정의할 수 있는 융통성(flexibility)을 갖기 때문에 다양한 목적으로 많이 활용되고 있다. XML 응용 중 최근에 가장 관심이 집중되고 있는 것은 웹 기반 XML 응용 문서들이다. 대표적인 응용 문서로는 CDF(Channel Definition Format), OSD(Open Software Description), WIDL(Web Interface Definition Language), UXF(UML eXchange Format) 등이 있다[1]. 이 중 CDF는 웹 사이트에서 웹 브라우저로 푸시(push)되는 데이터를 설명하는 포맷이며, OSD는 소프트웨어 패키지와 그들 간의 관계를 정의하기 위해 XML 기반의 형태로 나타낸 것이다[2]. 그리고, CDF 파일은 직접적으로 소프트웨어 의존 관계를 가리키기 위해 OSD를 포함할 수 있다[3].

한편, UML은 프로그램 개발 과정을 일관되게 진행하고 업무 담당자들 간의 의사소통을 용이하게 하여 프로젝트가 효율적으로 진행될 수 있는 공통적인 객체지향적 설계 언어이다[4]. UML로 XML 문서 구조를 표현하게 되면 XML 문서를 생성, 접근, 수정하는 XML 프로그램을 체계적이고 효율적으로 설계하며, 프로그램 개발 과정을 통합적으로 추진할 수 있다.

따라서, 본 논문에서는 OSD와 CDF 문서 인스턴스를 체계적이고 객체지향적으로 설계, 처리하기위해 UML 클래스 다이어그램으로 변환시키는 시스템을 제안하고자 한다.

본 연구를 통해 생성되는 클래스 다이어그램은 Rational Rose나 Cool : Joe와 같은 비주얼 객체 모델링 도구에서 사용 가능하며, 컴포넌트 다이어그램과 C++, 자바와 같은 객체지향 언어와 스키마로 쉽게 변환된다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구를 설명하고, 3장에서는 UML과 OSD, CDF 문서에 대해 설명한다.

※ 본 연구는 2002년도 경인여자대학 교내연구지원 연구비에 의해 수행되었음.
† 준 회 원 : 경인여자대학 컴퓨터정보기술학부 교수
논문접수 : 2003년 5월 16일, 심사완료 : 2003년 10월 2일

4장에서는 OSD나 CDF 문서를 입력받아 UML 클래스 다이어그램으로 변환시키는 시스템과 그 적용 결과를 제시한다. 그리고, 본 연구에서 제안한 변환 규칙들을 기존 연구들과 비교, 분석해 본다. 끝으로 5장에서는 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

XML 문서를 위한 표준화된 API를 제공하여 웹 문서에 접근하고 조작하기 위한 방법으로 DOM(Document Object Model)[1]이 있다. 이것은 문서의 논리적 구조를 정의하는 객체 기반 구조이기는 하나 트리 계층 구조의 형태이므로 클래스의 세부적인 정보인 애트리뷰트이나 집단화 관계 등을 제대로 표현하지 못한다.

이 외에 DTD에 대한 객체 모델링 연구는 XOMT[5]와 UML 클래스 다이어그램을 이용한 방법이 있다[6]. XOMT는 OMT(Object Modeling Technique)를 기반으로 확장한 형태이나, 객체지향 개념에 맞지 않는 일반화 관계 등을 사용하고 있다. 이에 비해, UML 클래스 다이어그램을 이용한 방법은 XML 링크부분을 지원할 수 있도록 DTD와 문서 인스턴스를 객체지향적으로 모델링하였다. 그러나, 이 방법은 일반적인 XML DTD와 문서 인스턴스에 맞게 설계되어 있으므로 특정한 목적으로 엘리먼트를 정의하여 사용하는 경우 충돌이 발생한다. 예를 들어, RDF(Resource Description Framework)의 경우 모든 엘리먼트가 클래스로 사상되는 것이 아니라 RDF 자원의 종류에 따라 클래스 대신 관련성을 나타내기도 하며[7], SMIL의 경우에도 동기 태그가 클래스로 사상되는 것이 아니라 클래스들 간의 메시지를 통해 동기화를 표현하도록 하고 있다[8]. 이와 같이 각 XML 응용의 경우 일괄적으로 적용되는 규칙 이외에 특정 응용 문서에서만 볼 수 있는 태그들이 존재한다. 특히, OSD의 경우 다른 응용에서 볼 수 없는 <IMPLEMENTATION>과 <DEPENDENCY>라는 태그가 존재하므로 이를 해석하여 UML 클래스 다이어그램의 구성 요소에 사상시키는 새로운 규칙이 필요하다. 따라서, 본 연구는 OSD와 CDF 문서의 엘리먼트를 각각 시맨틱적으로 분석하여 클래스 다이어그램을 생성하기 위한 시스템을 제안한다. 또한 지금까지 연구되어온 XML과 그 응용 문서들의 엘리먼트들을 UML 클래스 다이어그램의 구성요소에 사상시킨 규칙들을 비교, 분석한다. 이것은 각종 XML 응용 문서를 객체지향적으로 통합 모델링하고 스키마를 생성하여 OODBMS에 저장, 관리하기 위한 기반을 제공한다.

3. UML과 OSD, CDF

본 장에서는 UML의 관련성들, 클래스 다이어그램, 그리고, OSD와 CDF 문서의 엘리먼트에 대해 살펴본다.

3.1 UML

UML에 존재하는 다양한 관련성들과 UML에서 가장 중심이 되는 다이어그램으로서 클래스 다이어그램에 대해 살펴본다.

3.1.1 UML의 관련성

UML에서 사용되는 관련성은 크게 4가지로 꼽을 수 있으며, 그 내용은 다음과 같다[9].

① 의존(Dependency)

두 객체 간의 의미적인 관계로서, 한 객체의 애트리뷰트나 행위를 다른 객체가 상속받아서 영향을 줄 수 있는 관계이다. 표기는 다음과 같으며, 라벨을 가질 수 있다.

② 결합(Association)

객체들이 시간적, 공간적으로 연결된 집합을 결합 관계라고 한다. 라벨, 다중성(multiplicity), 역할 명(role name)을 표현할 수 있으며, 표기는 다음과 같다.

③ 집단화(Aggregation)와 컴포지션(Composition)

결합 관계의 특별한 종류로는 집단화와 컴포지션 관계가 있다. 집단화 관계는 전체와 부분간의 구조적인 관계를 나타내며, 표기는 다음과 같다.

컴포지션은 강한 집단화 관계라 하며, 상위 클래스의 생명 주기에 맞춰 하위 클래스가 생명주기를 갖는 것으로 이에 대한 표기는 다음과 같다.

DTD의 경우 선언된 엘리먼트는 엘리먼트 내용으로 구성된다. 본 논문에서는 이러한 특징을 상위 클래스와 하위 클래스들 간의 컴포지션 관계로 표현하며, 이에 대한 부가적인 정보로 다중성과 제한조건을 사용한다. 하나의 하위 클래스가 반복적으로 상위 클래스를 구성하는 경우 다중성으로 표현하며, 여러 개의 하위 클래스 중 하나만을 선택하고자 할 때, '{xor}' 관계를 부여한다.

④ 일반화(Generalization)/특성화(Specification)

일반화/특성화 관계는 일반화된 상위 객체의 애트리뷰트나 행위에 특수한 값(인스턴스)을 대응시켜서 특성화한다. 상위 객체는 특수화된 요소의 하위 객체로 세분화할 수 있는 관계이다.

⑤ 구현(Implementation)

분류자간의 의미적인 관계로, 한쪽 분류자는 다른 쪽 분류자가 수행하기로 되어있는 계약을 명세화한다. 이것은 인터페이스와 그것을 실현하는 클래스나 컴포넌트 사이 등에서 볼 수 있으며, 표기는 일반화 관계와 의존 관계를 절충한 형태이다. 다른 말로 실체화(realization) 관계라고도 한다.

3.1.2 클래스 다이어그램

클래스 다이어그램은 클래스들간의 관련성을 나타내는 것으로 객체 모델링에서 가장 공통적으로 쓰이는 다이어그램이다. 이것은 시스템의 정적인 설계 관점을 표현하는 것으로, 클래스는 동일한 애트리뷰트와 행위를 가진 객체들의 관계와 그들의 의미를 설명하는 객체들의 집합을 기술한 것이다. 클래스들간의 관련성에는 의존 관계, 결합 관계, 집단화 관계 그리고, 일반화 관계가 있다.

3.2 OSD(Open Software Description Format)

본 논문에서는 OSD의 엘리먼트들을 크게 주요(major) 엘리먼트와 보조(minor) 엘리먼트로 구별하여 기술한다[3, 10]

3.2.1 주요 엘리먼트

주요 엘리먼트와 이에 대한 설명은 <표 1>과 같다.

〈표 1〉 OSD의 주요 엘리먼트

주요 엘리먼트	설 명
SOFTPKG	소프트웨어패키지를 정의한다.
IMPLEMENT	소프트웨어 패키지의 구현을 기술한다.
DEPENDENCY	소프트웨어 일부분과 컴포넌트들간의 의존관계를 가리킨다.

3.2.2 보조 엘리먼트

보조 엘리먼트는 주요 엘리먼트에 대한 하위 엘리먼트들을 나타내며, 이에 대한 설명은 다음과 같다.

〈표 2〉 OSD의 보조 엘리먼트

보조 엘리먼트	설 명
TITLE	소프트웨어 이름을 나타낸다.
ABSTRACT	소프트웨어 부분에 대한 목적이나 본질을 짧게 기술한다.
LICENSE	라이센스 동의나 저작권을 고지한 위치를 나타낸다.
CODEBASE	소프트웨어 부분이 존재하는 위치를 알려준다. 반드시 한 개이상의 존재해야 한다.
OSVERSION	필요한 운영체제의 시스템 버전을 나타낸다.
PROCESSOR	필요한 프로세서(CPU)를 가리킨다.
LANGUAGE	소프트웨어 사용자 인터페이스에서 요구되는 자연어를 가리킨다.
VM	구현하는데 필요한 가상 기계를 나타낸다.
MEMSIZE	구현하는데 필요한 디스크 공간의 양을 나타낸다.
IMPLETYPE	구현의 타입을 가리킨다.

3.3 CDF(Channel Definition Format)

채널은 웹사이트를 구독하려는 독자들에게 정보를 보내주는 웹캐스팅이나 푸시기술에 사용되는 것을 말하며, 채널을 정의하는 XML 응용을 CDF라 한다.

3.3.1 주요 엘리먼트

주요 엘리먼트와 이에 대한 설명은 <표 3>과 같다.

〈표 3〉 CDF의 주요 엘리먼트

주요 엘리먼트	설 명
CHANNEL	채널을 정의한다.
ITEM	일반적으로 웹페이지를 나타낸다.
USERSCEDULE	스케쥴에 정의된 클라이언트 쪽의 사용자를 정의 한다.
SCHEDULE	반복되는 시간의 간격을 정의한다.
LOGO	채널이나 채널 아이템을 표현하기 위해 사용될 이미지를 정의한다.
TRACKING	어떻게 채널이 사용자 트래킹을 지원 할 수 있는지 가리킨다.
CATEGORYDEF	카테고리 특성에 맞는 컨테이너(container)를 제공한다.

3.2.2 보조 엘리먼트

보조 엘리먼트는 주요 엘리먼트에 대한 하위 엘리먼트들을 나타내며, 이에 대한 설명은 다음과 같다.

〈표 4〉 CDF의 보조 엘리먼트

주요 엘리먼트	설 명
ABSTRACT	페이지에 대해 요약 정리한다.
LOGTARGET	기록정보를 보낼 곳을 규정한다.
PURGETIME	기록정보가 유효한 시간의 수를 규정한다.
HTTP-EQUIV	기록 대상을 보내기 위해 HTTP-MIME 내에 핵심-값을 설정한다.
USAGE	다양한 채널의 사용방법을 설정한다.

4. 변환 시스템

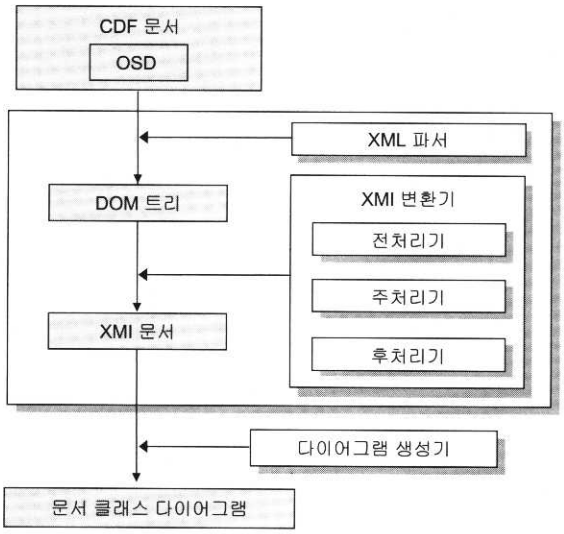
본 시스템은 OSD나 CDF 문서 인스턴스를 입력받아 사상 규칙과 알고리즘에 의해 UML 클래스 다이어그램으로 변환시킨다.

4.1 시스템 구성

OSD 문서 또는 OSD가 포함된 CDF 문서가 입력되었을 때, DOM 트리와 XMI 문서 형태를 통해 최종적으로 클래스 다이어그램을 생성하는 시스템의 구성도는(그림 1)과 같다.

4.1.1 XML 파서

대표적인 웹 기반 XML 응용인 OSD와 CDF 문서를 입력받아 문법상의 오류를 검사하여 DOM 트리를 생성한다. DOM 트리 형태의 객체들은 XMI 변환기의 입력이 된다.



(그림 1) UML 클래스 다이어그램 생성 시스템 구성도

4.1.2 XMI 변환기

XMI 변환기는 DOM 트리 형태의 객체를 입력으로 UML 클래스 다이어그램 정보를 나타내는 XML 형태의 XMI(XML Metadata Interchange)를 생성한다.

XMI는 분산 환경에서 모델링 도구와 메타데이타 저장소와의 교환을 쉽게 하기 위한 목적으로 OMG에서 RFC로 제시되었다. OMG의 모델링 표준으로 채택된 UML을 OMG의 메타데이타 저장소의 표준인 MOF(Meta Object Facility)로 옮기기 위한 목적으로 XMI는 사용된다. 현재 XMI는 Unisys, IBM, Oracle와 같은 벤더에서 UML 교환 형식으로 지원되고 있다. 대표적인 모델링 도구 중에 하나인 Rational Rose 2000*i* 에서는 Unisys에서 제공하는 Add-on 컴포넌트를 설치하면 XMI 형식으로 저장된 UML 모델을 읽어오거나 Rose에서 작성된 UML 모델을 XMI 형식으로 저장할 수 있다.

XMI 문서는 UML 클래스 다이어그램의 클래스 혹은 관련성 정보를 XML 문서 인스턴스로 구별하여 제시한다. XMI 변환기는 크게 전처리기, 주처리기, 후처리기로 구성되는데, 세부적인 기능은 다음과 같다. 전처리기는 DOM 트리 형태로부터 클래스 정보와 관련성 정보를 추출한다. 주처리기는 전처리기에서 추출된 클래스와 관련성 정보를 갖고 DOM 트리 형태를 재구성한다. 후처리기에서는 주처리기에 의해 생성된 DOM 트리 형태를 입력받아 XMI 문서를 생성한다. OSD나 CDF 문서를 변환시킨 XMI 문서는 클래스들과 이들의 컴포지션 관계 혹은 구현 관계를 나타낸다. 컴포지션 관계는 필요에 따라 다중성을 갖으며, 여러 개의 컴포지션 관계를 연결하여 '{or}' 관계를 갖도록 한다.

특히, OSD 문서에서는 컴포지션 이외에 구현 관계가 발생한다.

4.1.3 다이어그램 생성기

XMI 변환기에 의해 생성된 클래스 다이어그램 정보(XMI 문서)는 Rational Rose2000*i*와 같은 비주얼 객체 모델링 도구에 의해 클래스 다이어그램 형태로 디스플레이 된다.

4.2 문서 클래스 다이어그램

OSD나 CDF 문서에 대해 UML 클래스 다이어그램을 생성하는 모델링 규칙과 알고리즘, 그리고 적용 결과를 제시한다.

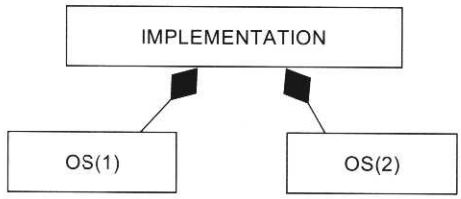
4.2.1 사상 규칙

OSD나 CDF 문서에 대해 UML 클래스 다이어그램의 엘리먼트와 애트리뷰트, 엘리먼트들 간의 관계에 관한 규칙을 정의하면 다음과 같다.

① 엘리먼트

시작 태그가 되는 엘리먼트에 대한 규칙들을 정의하면 다음과 같다.

[규칙 1] 시작 태그가 되는 엘리먼트는 클래스가 된다.
이 때, 같은 타입의 클래스가 반복 생성되는 경우 각 클래스 이름에 순차 번호를 붙여 이를 구별해 준다.
예) <IMPLEMENTATION>
<OS> … </OS>
<OS> … </OS>
<IMPLEMENTATION>

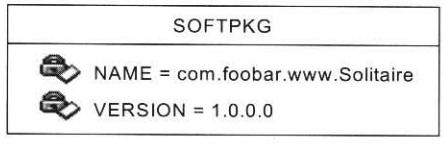


(그림 2) [규칙 1]의 클래스 다이어그램

② 애트리뷰트

엘리먼트 내의 애트리뷰트와 그 값에 관한 규칙은 다음과 같다.

[규칙 2] 엘리먼트 태그 내에 애트리뷰트와 값은 해당 클래스의 애트리뷰트와 값이 된다.
예) < SOFTPKG NAME = "com.foobar.www.Solitaire" VERSION = "1,0,0,0" >



(그림 3) [규칙 2]의 클래스 다이어그램

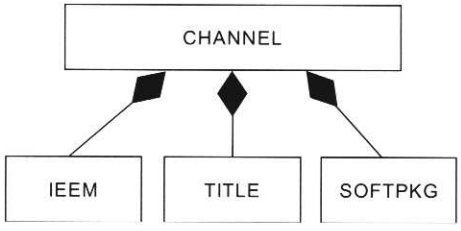
③ 엘리먼트들 간의 관계

엘리먼트와 그 안에 포함된 엘리먼트들간의 관계는 다음

과 같이 정의한다.

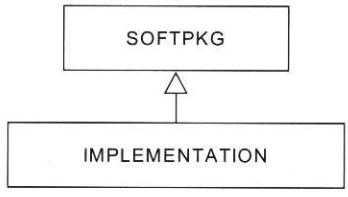
[규칙 3] 엘리먼트 태그 안에 오는 엘리먼트는 클래스와 하위 클래스들 간의 집단화 관계가 된다.
단, <IMPLEMENTATION> 태그는 구현 관계, <DEPENDENCY> 태그는 의존 관계가 된다.

예 1) <CHANNEL … >
<ITEM> … </ITEM>
<TITLE> … </TITLE>
<SOFTPKG> … </SOFTPKG>
</CHANNEL>



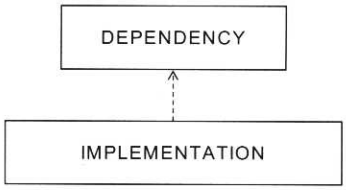
(그림 4) [규칙 3](집단화 관계)의 클래스 다이어그램

예 2) < SOFTPKG … >
< IMPLEMENTATION >
…
< /IMPLEMENTATION >
< /SOFTPKG >



(그림 5) [규칙 3](구현 관계)의 클래스 다이어그램

예 3) <IMPLEMENTATION>
< DEPENDENCY >
…
</DEPENDENCY>
</IMPLEMENTATION>



(그림 5) [규칙 3](구현 관계)의 클래스 다이어그램

4.2.2 모델링 함수

UML 클래스 다이어그램에서 각 클래스의 오퍼레이션리스트 부분에 삽입(insert)되는 것으로, 클래스 다이어그램의 구조 파악을 용이하게 해준다.

모델링 함수를 정의하기 위한 가정과 정의는 다음과 같다.

[가정 1] 주요 클래스의 타입 집합 T={SOFTPKG, IMPLEMENTATION, DEPENDENCY, CHANNEL, ITEM} 이다.

[가정 2] 클래스 다이어그램은 (S, p)의 쌍으로 이루어진다. 단, S는 집합이고, 함수 $p : S \rightarrow S : \forall t \in S$이다.

위의 [가정 2]에서 S의 클래스 t는 아래의 특성을 만족한다[5].

- $p(t) = t$, t는 S의 클래스이며, 클래스 다이어그램의 루트(root)이다.
- $\forall x \in S,\ \exists k \in N,\ p^k(x) = t$

① 상위클래스, 하위클래스

상위클래스와 하위클래스를 나타내는 함수를 정의하면 다음과 같다.

[정의 1] 함수 parent, child : $T \rightarrow parent(T) : \forall_t \in T$는 각각 상위클래스와 하위클래스를 나타낸다. 단, $t_2 = child(t_1)$이면 $t_1 \neq t_2$이다.
예) <OS VALUE = "WinNT">
<OSVERSION VALUE = "4,0,0,0"/ ></OS>
⇒ parent(OSVERSION) = OS, child(OS) = OSVERSION

[정의 2] 함수 imple_no : $T \rightarrow Integer : \forall_t \in T$, c(t) = IMPLEMENTATION일 때, 구현의 개수를 표시한다.
예) <SOFTPKG NAME = "com.foobar.www.Solitaire" VERSION = "1,0,0,0">
<IMPLEMENTATION>
…
</IMPLEMENTATION>
<IMPLEMENTATION>
…
</IMPLEMENTATION>
</SOFTPKG>
⇒ parent(IMPLEMENTATION) = SOFTPKG
child(SOFTPKG) = IMPLEMENTATION
imple_no(SOFTPKG) = 2

[정의 3] 함수 depen_no : $T \rightarrow Integer : \forall_t \in T$, c(t) = DEPENDENCY일 때, 의존 관계의 개수를 표시한다.
예) <IMPLEMENTATION>
<DEPENDENCY>
…
</DEPENDENCY>
</IMPLEMENTATION>

⇒ depen_no(IMPLEMENTATION) = 1

[정의 4] 함수 item_no : T → Integer : $\forall_t \in T$, c(t) = ITEM 일 때, 의존 관계의 개수를 표시한다.

<CHANNEL HREF = …>
<ITEM HREF … > … </ ITEM>
<ITEM HREF … > … </ ITEM>
</CHANNEL>
⇒ item_no(CHANNEL) = 2

② 애트리뷰트

클래스에 속하는 애트리뷰트를 정의하면 다음과 같다.

[정의 5] 함수 attr : T → A : $\forall t \in T$로 클래스에 속한 모든 애트리뷰트를 표시한다.

예) <SOFTPKG NAME = "com.foobar.www.Solitaire" VERSION = "1,0,0,0">
⇒ attr(SOFTPKG) = {NAME, VERSION}

4.2.3 모델링 알고리즘

OSD나 CDF 문서를 입력하여 XMI 문서를 생성하는 알고리즘은 다음과 같다.

```
입력 : OSD/CDF 문서 인스턴스
출력 : XMI 문서
begin
{
 루트 클래스 생성
 // SOFTPKG, CHANNEL 클래스 생성
 for ( 애트리뷰트 리스트 개수 )
    애트리뷰트와 값 삽입
 멤버 함수 삽입

 for ( 시작 태그 개수 )
 {
   클래스 생성
   for ( 애트리뷰트 리스트 개수 )
        애트리뷰트와 값 삽입
   멤버 함수 삽입
   if ( < IMPLEMENTATION > )
     {
       구현 관계 형성
       for ( 시작 태그 개수 )
       {
        클래스 생성
        for ( 애트리뷰트 리스트 개수 )
             애트리뷰트와 값 삽입
        멤버 함수 삽입
        if (<DEPENDENCY>)
          {
            의존 관계 형성
            for ( 시작 태그 개수 )
            {
             클래스 생성
             for ( 애트리뷰트 리스트 개수 )
                  애트리뷰트와 값 삽입
             맴버 함수 삽입
            }
          } // 의존 관계
        else 컴포지션 관계 형성
       }
     } // 구현 관계
   else 컴포지션 관계 형성
  }
 }
end ;
```

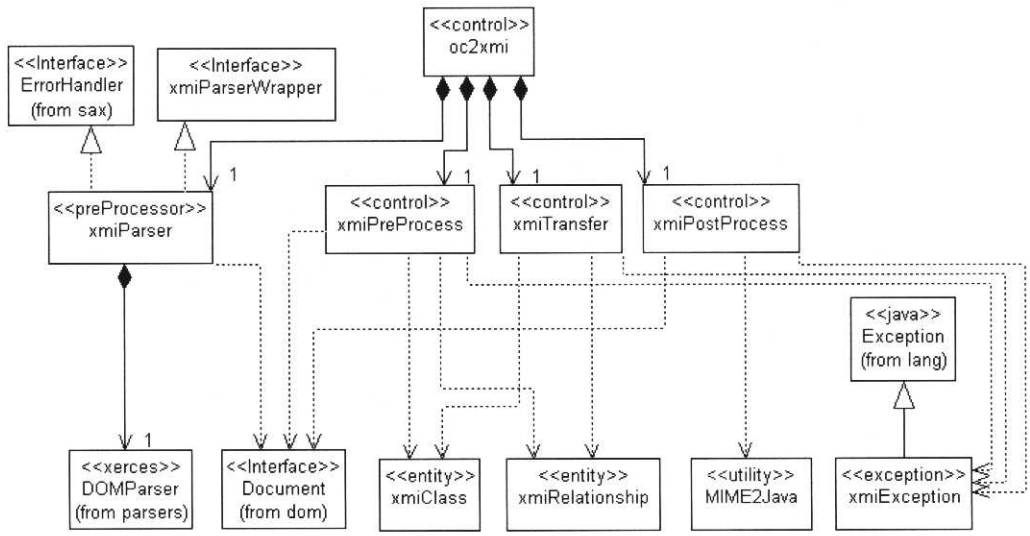
4.3 구현

본 절에서는 4.2.3 알고리즘의 구현 시스템 및 자바 클래스들을 소개하고, 적용 결과를 제시한다.

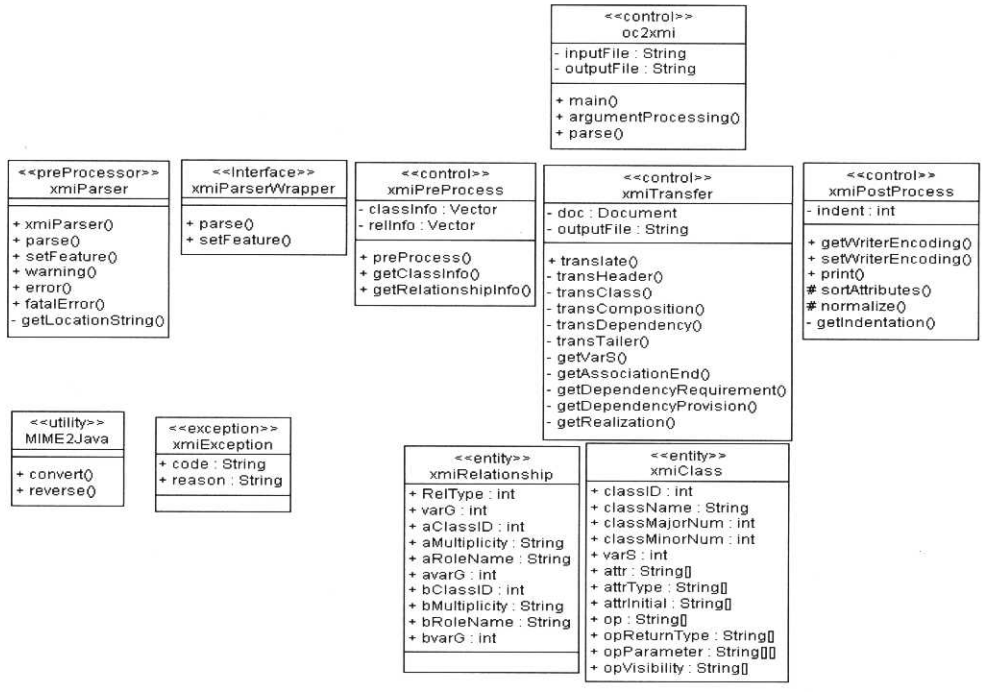
4.3.1 구현 환경 및 자바 클래스

사용 언어는 JDK 1.4이며, XML 파서는 Xerces1.4.4을 사용하였다. 구현을 위해 사용된 자바 클래스들에 대한 클래스 다이어그램과 세부적인 형태는 (그림 7), (그림 8)과 같다.

(그림 7)의 각각의 클래스들의 간략한 설명은 <표 5>와 같다.



(그림 7) XMI 문서 생성을 위한 클래스 다이어그램



(그림 8) (그림 7)의 주요 클래스들

xmiClass 클래스와 xmiRelationship 클래스는 XMI 주처리기에서 OSD나 CDF 문서의 내용을 4.2.1절 사상 규칙에 의해 생성된 클래스와 클래스간의 관련성을 내용을 저장하는 엔터티클래스이다.

〈표 5〉 각 클래스들의 기능

클래스 명	설 명
xmiParserWrapper	Xercess XML 파서와 래핑하기 위한 인터페이스
xmiPreProcess	XMI 전처리기 클래스
xmiTransfer	XMI 주처리기 클래스
xmiPostProcess	XMI 후처리기 클래스
oc2xmi	main() 구동 클래스
MIME2Java	MIME character 이름을 자바 인코딩 이름으로 매칭한 정보를 저장한 클래스
xmiClass	클래스 정보를 저장하는 엔터티 클래스
xmiRelationship	클래스들간의 관련성 정보을 저장하는 엔터티클래스
xmiException	oc2xmi 시스템의 예외 클래스
xmiParser	oc2xmi의 XML 파서 클래스

xmiClass 클래스는 클래스, 애트리뷰트, 연산에 대한 내용을 갖는다. 클래스에 대한 내용으로는 클래스 식별자, 클래스 이름, 그리고 클래스 이름이 중복으로 나올 때 사용되는 버전 번호가 포함되어 있고, 클래스 애트리뷰트에 대한 내용으로는 클래스 애트리뷰트 이름, 애트리뷰트 타입, 애트리뷰트 초기 값, 애트리뷰트의 가시성이 포함된다. 클래스 연산에 대한 내용으로는 연산 이름, 반환 타입, 입력 파라미터, 연산 가시성이 포함되어 있다.

xmiRelationship 클래스에는 관련성의 타입과 관련성의 양쪽 클래스의 참조, 양쪽 클래스의 식별자, 양쪽 클래스의 다중성, 양쪽 클래스의 역할이름이 포함된다.

xmiException 클래스는 java.lang.exception을 상속받아서 XMI 문서 생성에서 발생하는 에러에 대해 예외를 발생할 때 사용한다.

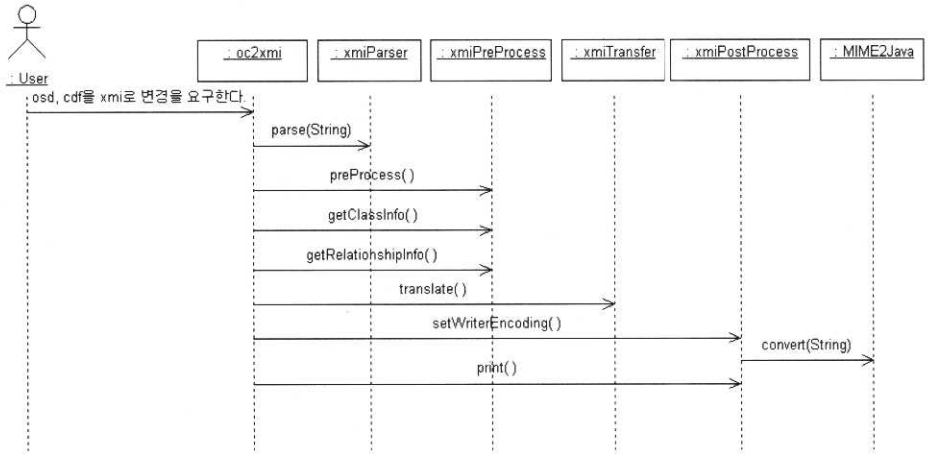
oc2xmi 클래스는 main()이 포함된 클래스로써 사용자가 입력한 인수를 처리하고 입력받은 OSD, CDF 파일에 대해 xmiParser 클래스에게 파싱을 요구한다. OSD, CDF 파일이 에러없이 파싱된 경우 oc2xmi 클래스는 OSD와 CDF 파일의 내용을 DOM 구조로 가지게 된다. oc2xmi클래스는 이 DOM 구조를 가지고 xmiPreProcess클래스에게 XMI 전처리를 요구한다.

xmiPreProcess 클래스는 DOM 구조를 탐색하면서 xmi Class, xmiRelationship 엔터티클래스에 4.2.1절의 사상 규칙에 의해 생성되는 클래스와 클래스간의 관련성에 대한 정보를 저장한다.

xmiPreProcess 클래스가 전처리를 끝내면 oc2xmi 클래스는 xmiClass와 xmiRelationship 클래스를 요구를 하고 xmiTransfer 클래스에게 XMI 주처리를 요구한다.

xmiTransfer 클래스는 xmiClass와 xmiRelationship 엔터티 클래스의 내용을 탐색하면서 XMI 문서 구조를 가지는 DOM 구조를 생성한다.

이 XMI DOM 구조가 생성이 완료된 후에 oc2xmi 클래스는 xmiPostProcess클래스에게 XMI DOM 구조를 파일로



(그림 9) XMI 문서 생성을 위한 시컨스다이어그램

저장을 요구한다. xmiPostProcess는 XMI 파일을 저장하는데 자바 인코딩 이름을 XML에서 사용하는 MIME character 타입 이름으로 출력해야 함으로 MIME2Java 클래스와 의존 관계를 가진다.

(그림 9)에 XMI 문서 생성을 위한 시컨스 다이어그램이 표현되어 있다.

4.3.2 구현 결과

다음은 OSD 문서를 입력하여 XMI 문서를 생성하고, 이를 클래스 다이어그램 생성기에 적용한 결과를 제시한다.

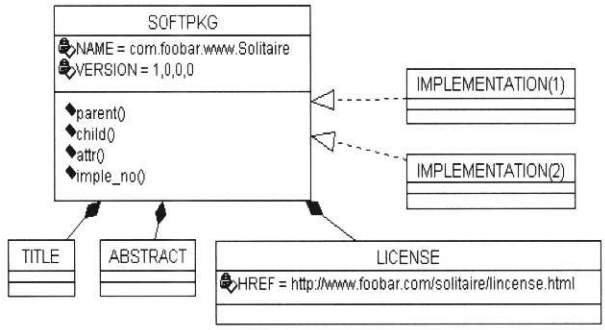
① 문서 인스턴스

OSD 문서 인스턴스를 입력으로 했을 때, 생성되는 UML 클래스 다이어그램은 다음과 같다.

```
<SOFTPKG NAME = "com.foobar.www.Solitaire"
                                    VERSION = "1,0,0,0" >
        <TITLE> Solitaire </TITLE>
        <ABSTRACT> Solitaire by FooBar Corporation
        </ABSTRACT>
        <LICENSE HREF = "http://www.foobar.com/
                        solitaire/license.html"/>
        <IMPLEMENTATION>
            <OS VALUE = "WinNT">
            <OSVERSION VALUE = "4,0,0,0"/></OS>
            <OS VALUE = "Win95"/>
            <PROCESSOR VALUE = "x86"/>
            <LANGUAGE VALUE = "en"/>
            <CODEBASE HREF = "http://www.foobar.org/solitaire.
                                cab"/>
        </IMPLEMENTATION>
        <IMPLEMENTATION>
            <IMPLTYPE VALUE = "Java"/>
            <CODEBASE HREF = "http://www.foobar.org/solitaire.
                                jar"/>
        <DEPENDENCY>
            <CODEBASE HREF="http://www.foobar.org/cards.
                                osd"/>
        </DEPENDENCY>
    </IMPLEMENTATION>
</SOFTPKG>
```

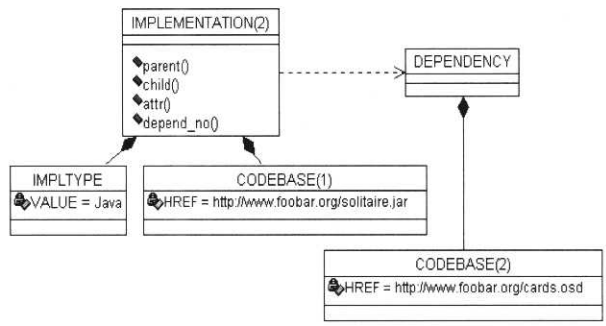
② 문서 인스턴스 클래스 다이어그램

입력된 OSD 문서 인스턴스에 대한 UML 클래스 다이어그램은 (그림 10)과 같다.



(그림 10) SOFTPKG의 클래스 다이어그램

(그림 10)에서 클래스 IMPLEMENTATION(2)에 대한 세부적인 그림은 (그림 11)과 같다.



(그림 11) IMPLEMENTATION(2)의 클래스 다이어그램

IMPLEMENTATION(2) 클래스에는 의존 관계를 나타내는 DEPENDENCY 클래스를 포함한다.

4.3.3 비교 분석

본 연구의 CDF와 OSD 문서에 대한 모델링을 기존에 제

〈표 6〉 XML 각 응용 모델링 규칙 비교

UML	XML	RDF	SMIL	WIDL/CDF	OSD
클래스	모든 엘리먼트 태그	<rdf : Description…>	미디어태그, 하이퍼링크태그, 선택태그	모든 엘리먼트 태그	모든 엘리먼트 태그
집단화(혹은 컴포지션) 관계	일반적인 태그들 간의 관계	일반적인 태그들 간의 관계	없 음	일반적인 태그들 간의 관계	일반적인 태그들간의 관계
구현 관계	없 음	없 음	없 음	없 음	<SOFTPKG …> <IMPLEMENTATION>
의존 관계	< ELINK … > < LOCATOR… >	없 음	없 음	없 음	<IMPLEMENTATION> <DEPENDENCY>
일반화 관계	기본타입클래스와의 관계	< rdf : subClassOf…>	기본타입클래스와의 관계	기본 타입 클래스와의 관계	기본 타입 클래스와의 관계

안되었던 다른 XML 문서 인스턴스에 대한 모델링과 비교해보면 <표 6>과 같다.

일반적으로 XML 문서 인스턴스에서 시작 태그로 사용되는 엘리먼트 태그들은 UML의 클래스로 사상이 된다. 단, RDF 자원과 SMIL은 특수한 경우로, RDF 자원에서 rdf : Description만이 클래스가 되고, 다른 시작태그는 관련성이나 주석이 된다. SMIL에서는 동기 태그 등은 관련성을 결정하므로 클래스를 생성하지 않는다.

그리고, XML 문서 인스턴스에서 SMIL의 경우를 제외하고 대부분 태그들 간의 관계는 집단화(혹은 컴포지션) 관계이다. 예외인 경우는 XML에서 확장 링크를 사용하는 경우에 의존 관계이며, OSD에서 <IMPLEMENTATION>과 <DEPENDENCY>를 사용하는 경우 각각 구현 관계와 의존 관계가 된다.

또한, 일반화 관계는 클래스들이 기본 타입 클래스로부터 상속을 받는 경우가 발생하며, RDF 자원의 경우 rdf : sub ClassOf 태그로 제한된다. 따라서, 본 연구의 OSD 객체 모델링은 다른 연구에서 볼 수 있는 컴포지션과 일반화 관계 이외에 의존 관계와 구현 관계를 명시한 모델링 방법이다.

〈표 7〉 XML 각 응용 모델링 함수 비교

멤버함수 종류	XML	RDF	SMIL	WIDL/CDF	OSD
상위클래스	O	O	O	O	O
하위클래스	O	O	O	O	O
애트리뷰트	O	O	O	O	O
링크되는 클래스	O	X	O	X	X
연결자종류	O	O	X	X	O
동시발생 클래스	X	X	O	X	X
순차적 발생 클래스(혹은 하위 클래스 순서)	O	O	O	X	X
구현클래스 개수	X	X	X	X	O
의존클래스 개수	X	X	X	X	O

그리고, <표 7>은 각 XML 응용에서 제안하는 모델링 함수 종류를 비교한 것이다. 각각 <표 6>에서 정의한 규칙으로부터 파생되는 함수들이다. 본 연구의 CDF 경우 WIDL과 같은 모델링 함수를 갖는다. OSD의 경우는 다른 응용에 비해 링크 부분과 클래스 발생 순서에 관련된 함수는 없으나, 구현과 의존 관계를 표현하기 위한 함수가 존재한다.

5. 결론 및 향후 연구 과제

본 논문은 웹 문서에 대한 채널을 정의하는 CDF와 직접적인 접근을 나타내는 인터페이스를 기술하기 위한 OSD에 대한 객체 모델링을 제안하였다. CDF 문서에 대한 객체 모델링은 WIDL의 객체 모델링 방법과 같으나, OSD의 경우 구현과 의존 관계 등을 위한 새로운 모델링 방법이 필요하다. 이와 같은 OSD와 CDF의 객체 모델링은 객체지향적인 문서 처리를 위한 효율적인 환경을 제시하며, 복잡한 XML 문서를 이해하기 쉽게 표현할 것이다. 특히, OSD와 CDF 클래스 다이어그램은 모델링 함수를 포함한 클래스 다이어그램을 제시하고 있으므로 문서의 구조를 통합, 변형을 용이하게 한다.

향후 연구 과제로는 OSD와 CDF의 객체 모델링을 다른 XML 응용 문서의 객체 모델링과 통합하여 데이터베이스를 구축하는 것이다.

참 고 문 헌

[1] Natanya Pitts-Moultis, Cheryl Kirk, "XML Black Book," The Coriolis Group Inc., 1999.

[2] Elliotte Rusty Harold 저, 김용권 역, "XML Bible", 정보문화사, 2000.

[3] Marimba, "OSD-Describing Software Packages on the Internet," 1998, http://www.marimba.com/products/whitepapers/osd-wp.html.

[4] 김채미, 최학열, 김심석, "전문가와 함께하는 XML Camp", 마이트프레스, 2001.

[5] 박인호, 한에노, 정은주, 김은정, 배종민, 강현석, 김완석,

"XOMT : SGML DTD 설계를 위한 객체 다이아그래밍 기법", 정보과학회논문지(C), 제3권 제3호, pp.228-237, 1997.

[6] 채원석, 하 얀, 김용성, "UML 클래스 다이어그램을 이용한 XML 문서 구조 다이어그래밍", 정보처리학회논문지, 제6권 제10호, pp.2670-2679, 1999.

[7] 이미경, 하 얀, 김용성, "RDF 스키마에서 UML 클래스 다이어그램으로의 변환", 정보처리학회논문지, 제7권 제1호, pp.29-40, 2000.

[8] 채원석, 하 얀, 김용성, "UML 사용사례 및 순서 다이어그램을 이용한 SMIL 문서 동기화", 정보과학회논문지(소프트웨어 및 응용), 제27권 제4호, pp.357-369, 2000.

[9] Grady Booch, James Rumbaugh, Ivar Jacobson 저, 임춘봉, 신인철, 심재철 역, "UML 사용자 지침서", 인터비젼, 1999.

[10] W3C, "The Open Software Description Format(OSD)," http://www.oasis-open. org/cover/osd 970813 NOTE.html. 1997.

[11] E. Akpotsui, V. Quint, C. Roisin. "Type Modelling for Document Transformation in Structured Edition Systems," Mathematical and Computer Modelling, Vol.25, No.4, pp.1-19, 1997, http://www.oasis-open.org/ cover/.

[12] James rumbaugh, Ivar Jacobson, Grady Booch, "The unified modeling language reference manual", Addison Wesley Longman Inc., 1999.

하 얀

e-mail : white@kic.ac.kr

1992년 덕성여자대학교 전산학과(학사)

1994년 이화여자대학교 전자계산교육전공(석사)

2000년 전북대학교 대학원 전산통계학과(이학박사)

2000년~2001년 중앙대학교 정보통신연구소 연구전담교수

2001년~현재 경인여자대학 컴퓨터정보기술학부 전임강사

관심분야 : XML 응용, 객체지향 모델링, 컴포넌트 모델링, 애니메이션, 멀티미디어