UML 개발 산출물을 위한 관계 및 내용 기반 검색 시스템

(A Retrieval System for UML Development Artifacts based on Relationship and Content)

전 세 길 † 나 연 묵 ††
(Segil Jeon) (Yunmook Nah)

요 약 본 논문에서는 UML을 기반으로 한 객체지향 시스템 개발에서 발생하는 멀티미디어 산출물들을 저장, 관리하기 위한 객체 저장소의 산출물 관리 기법을 제시하였다. 객체 관리 기능 모듈은 상용 객체 관계 DBMS인 IUS를 이용하여 구현하였다. 산출물 객체의 표현 및 데이타의 공유를 위해 UML 다이아그램 산출물들을 XML 문서 형태로 표현하고, eXcelon XML 데이타 서버를 이용하여 XML 문서처리 시스템을 구현하였다. 산출물 객체의 관계 관리 기법과 UML 다이아그램에 대한 XML 문서화 기법은 UML을 기반으로 하는 CASE 도구의 개발에 활용될 수 있다.

Abstract In this paper, we proposed techniques to handle artifacts of the object repository to store and manage multimedia artifacts produced by UML-based object-oriented development process. The object management modules are implemented using the commercial object-relational DBMS, IUS. Also, we transformed the UML diagram artifacts into XML documents for the sharing and representation of artifacts and implemented the prototype XML document system using eXcelon XML data server. The research results can be utilized for the development of UML-based CASE tools.

1. 서 론

현재의 정보 시스템은 컴퓨터 기술의 발전과 인터넷의 보급으로 대량의 컴퓨터가 네트워크에 연결되어 있으며, 수시로 자료의 교환이 이루어지고 있다. 또한, 교환하는 자료의 형태도 단순 텍스트 형태의 문서, 그래픽, 음성, 동영상, 웹 문서, 소스코드 등 멀티미디어 자료의 형태를 취하고 있다.

객체지향 패러다임의 보급으로 인해 객체지향 소프트웨어 개발을 위한 객체 지향 소프트웨어 공학 방법론인 Rumbaugh의 OMT(Object Modeling Technique), Jacobson의 Objectory, Booch의 OOA/OOD, Rumbaugh, Jacobson, Booch 방법론의 통합 형태인 UML(Unified Modeling Language) 등이 제시되었다[1, 2, 3].

이러한 객체지향 기술 개발 단계에서의 멀티미디어 산출물을 재사용하기 위해서는 멀티미디어 자료에 대한 저장, 검색이 가능한 저장소가 필요하며, 특히 UML을 기반으로 하는 객체지향 기술 개발에서는 UML 메타모델을 지원해야 한다[4, 8, 9].

일반적으로 객체의 이전 버전이나 대안적인 버전을 접근하기 위한 버전 정보에 대한 관리는 엔지니어링 디자인 응용(CAD/CAM)과 사무 자동화(문서관리) 등의 분야에서 사용되고 있으며, 대부분의 객체지향 데이타베이스에서도 객체 관리를 위한 핵심 기능으로 사용되고 있다[18, 20].

또한 이러한 응용에서는 같은 객체의 다른 버전을 추적하기 위해서 다른 객체와의 버전 관계성을 표현하며, 이를 위한 자동 버전 관리는 매우 유용하다. 그러나, 소프트웨어 개발을 지원하는 자동화 도구인 CASE 도구나 기타 객체지향 개발을 지원하는 멀티미디어 객체 저장소에서는 저장 대상 산출물 객체에 대해 버전 관리

· 이 논문은 과학기술부의 핵심S/W개발 과제(98-NS-01-08-A-27)의 지원과 한국과학재단의 특정기초연구과제(98-0102-06-01-3)의 일부 지원에 의한 것임
† 학생회원 : 단국대학교 컴퓨터공학과
sgjeon@dankook.ac.kr
†† 종신회원 : 단국대학교 컴퓨터공학과 교수
ymnah@dankook.ac.kr
논문접수 : 2000년 4월 12일
심사완료 : 2000년 11월 27일

기법을 적용한 사례가 거의 없으며, 산출물 객체의 버전 관리를 위한 자동화가 미흡한 상태이다.

대부분의 인터넷을 기반으로 한 개발에서는 여러 개발자, 플랫폼 하에서 자료교환이 이루어지며, 이때 서로 다른 도구들간의 자료교환을 위해서는 표준화된 형태의 자료 형식이 필요하게 된다. 이와 관련하여, 최근 웹 상에서의 복잡한 멀티미디어 형태의 자료는 표준화되고, 일관된 형식으로 정의하고 전송할 수 있는 W3C의 XML(eXtensible Markup Language) 문서의 형태로 표현되고 있다[5].

현재 UML 데이타 모델을 XML 문서 형태로 변환하는 기술은 UXF(UML eXchange Format)와 같은 기술이 있으며, OMG의 UML 메타모델의 추상구문을 기반으로 작성되어 있다[22]. UXF는 메타 모델을 전체적으로 반영하지 않았으며, 시퀀스 다이아그램에 대한 정의도 빠져있다. 또한 UXF에서는 UML 데이타 모델에 대한 DTD와 변환된 XML 문서 저장을 위한 저장소에 대한 기술이 언급되어 있지 않다.

Rational의 Rose나 UML을 지원하는 CASE 도구들은 객체 지향 개발 과정 중의 다양한 산출물들을 파일에 저장한다. 이러한 방법은 데이타베이스 기술의 결여로 인해 산출물의 필요한 객체에 대한 조건 검색이 불가능하고 동시적인 공유도 불가능하다. 비교적 최근에 제시된 방법인 Microsoft의 Repository는 Microsoft의 관계 DBMS인 SQL Server를 이용하여 산출물에 대한 동시 공유와 효율적인 재사용을 지원하고 있다[13, 16, 17].

그러나, 이 제품은 Microsoft 플랫폼에 종속적이며, 현재 Visual Basic 개발 환경을 위한 컴포넌트(component)의 지원과 탐색을 위주로 지원하고 있어 C++이나 Java와 같은 다양한 개발 환경을 지원하지 못하고 있고, 다양한 산출물을 모두 관리하지 못하는 단점이 있다.

본 논문에서는 웹 상에서 UML 모델을 기반으로 하는 객체지향 소프트웨어 개발 시에 발생하는 멀티미디어 형태의 산출물인 텍스트 문서, XML 문서, 그래픽 형태의 산출물 등을 저장하고 관계 및 내용기반 검색을 지원하는 산출물 객체 저장소를 구축한다. 이를 위해, 제반 메타 정보에 대한 메타 스키마를 작성하고, 관계 관리(relationship management), 버전 관리(version management) 등의 저장소 기능을 구현한다. 이 메타 스키마와 저장소의 산출물 관리 기능에 의해 산출물 객체의 저장 및 관계기반 검색이 가능해 진다. 그리고, 그래픽 형태의 UML 모델 정보를 구조적인 텍스트 문서 형태로 기술하는 표준 포맷을 제공하고, 웹 상의 서로 다른 플랫폼 하에서 도구들간의 정보 교환 기능을 제공하기 위해 UML 다이아그램을 XML 문서 형태로 기술하여 XML 문서처리 저장소에 저장해 멀티미디어 산출물 객체 자체에 대한 내용 기반 검색이 가능하게 된다.

본 논문의 구성은 다음과 같다. 2장에서는 저장 대상 산출물의 기반이 되는 UML 기반 객체지향 기술개발에서 발생하는 산출물에 대해 설명한다. 3장은 저장 산출물에 대한 관계 기반 검색 기술에 대해 설명하고 4장에서는 저장 산출물에 대한 내용 기반 검색 기술에 대해 설명한다.

5장은 UML 산출물 객체의 관계 및 내용 기반 검색 시스템의 구현에 대해 설명한다. 마지막으로 6장에서는 결론과 향후 연구 방향을 제시한다.

2. UML 개발 과정의 산출물

UML(Unified Modeling Language)은 기존의 Booch 방법론, Rumbaugh 등의 OMT(Object Modeling Technique), Jacobson의 OOSE 방법론 등을 연합하여 만든 모델링 개념의 공통 집합으로, 객체지향적 분석/설계 방법론의 표준 방법론화를 목표로 하고 있다[8, 9, 10].

현재 UML 표준안은 UML의 구성 요소가 된 이전 방법론들을 이미 사용하여 왔던 많은 대규모 소프트웨어 기업들(예를 들어 Oracle, MS, HP, TI, Unisys, DEC 등)로부터 지원을 받고 있다.

저장대상이 되는 산출물 객체의 구성을 대략적으로 보면 그림 1과 같으며, 정적인 구조를 나타내는 클래스 다이아그램, 동적인 구조를 나타내는 시퀀스 다이아그램, 산출물 객체에 대한 설명 문서 등으로 구성된다. 구현코드는 CORBA, DCOM 기반의 IDL 코드, COM 인터페이스, ActiveX Control, Java 코드 등의 실제 구현 코드이다.

그림 형태의 클래스 다이아그램과 시퀀스 다이아그램은 XML 문서 형태로 변환될 수 있으며, 이 XML 문서

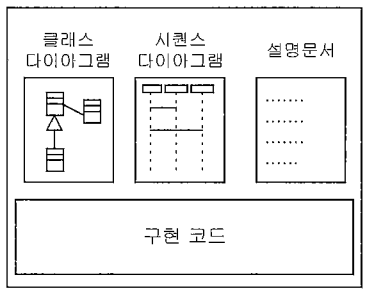


그림 1 멀티미디어 산출물의 구성

도 저장소의 저장 및 검색 대상이 된다. 객체 저장소에 저장되는 대상 산출물은 다음과 같이 산출물 객체의 구성 형태와 산출물 객체를 구성하는 구성 요소에 따라 분류할 수 있다.

(1) 산출물 객체의 구성 형태에 따른 분류

산출물 객체의 타입은 단일 산출물 객체(single object)와 집단 산출물 객체(aggregation object)로 나누어진다.

(2) 산출물 객체의 구성 내용에 따른 분류

본 논문의 대상인 UML 기반의 객체지향 산출물 객체는 ①, ②에서와 같이 객체지향 분석/설계 단계 산출물 객체와 구현 산출물 객체로 분류할 수 있다.

① 분석/설계 단계의 산출물 객체

다이아그램, XML 문서, 메뉴얼 문서

② 구현 단계의 산출물 객체

소스 코드

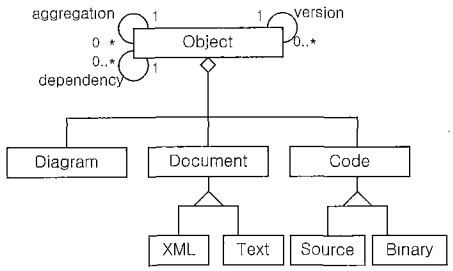


그림 2 산출물 객체의 UML 클래스 다이아그램

UML을 기반으로 하는 객체지향 개발의 산출물 객체는 객체의 정적인 구조와 동적인 행위를 나타내는 다이아그램과 설명문서, 다이아그램에 대한 XML 문서, 구현코드로 구성된다. 산출물 객체의 실제 구현코드는 텍스트 형태의 코드와 실행 가능한 이진화일 형태의 코드로 구분할 수 있다. 이러한 UML 산출물 객체의 구성 요소와 구성 요소간의 관계를 UML 클래스 다이아그램 형태로 그려보면 그림 2와 같다.

그림 2에서와 같이 객체들 간에는 집단, 종속 관계를 가지며 한 객체는 여러 개의 버전화된 객체를 가질 수 있다.

3. UML 산출물에 대한 관계 정보 관리

산출물 객체 저장소는 산출물 객체간의 다양한 관계를 표현할 수 있어야 한다. 즉, 산출물 객체간에 발생하는 종속 관계, 단일 산출물 객체간의 포함관계, 단일 산출물 객체와 집단 산출물 객체간의 포함 관계 등을 지원해야 한다.

특정 객체는 시간이나 개발 목적에 따라 변화할 수 있으며 이러한 관계에 의해 변화 생성되는 객체를 버전 객체라 한다. 또한, 저장소내의 데이타는 버전 관계에 의해 특정 객체의 이전 버전, 원본 버전, 최종 버전의 객체를 검색 및 참조할 수 있어야 한다.

(1) 산출물 객체간의 종속 관계

산출물 객체 저장소내의 모든 산출물 객체들은 서로를 참조할 수 있으며, 객체 저장소는 각 단일 산출물 객체의 종속(dependency) 관계를 나타낼 수 있어야 하며, 산출물 객체의 삽입, 삭제 시에 변동 사항을 종속 관계 정보에 반영해 주어야 한다.

(2) 산출물 객체간의 포함 관계

집단(aggregation) 산출물 객체는 여러 개의 산출물 객체를 조립한 것이며, 객체 저장소는 이러한 산출물 객체의 조립 과정을 지원해야 한다. 또한 저장소는 집단 산출물 객체에 포함된 하위 객체의 집합을 유지하여야 한다.

(3) 산출물 객체의 버전 관계

버전은 사용자 객체의 시간에 따른 변화나 특정 목적을 가지는 새로운 형태의 객체라고 할 수 있다. 시간의 순서에 따라 생성되는 버전을 순차 버전(linear version)이라고 하며, 여기서 시간은 산출물 객체가 생성된 시간과 실제로 데이타베이스에 저장된 시간으로 볼 수가 있는데, 본 논문에서는 산출물 객체가 개발자에 의해 새로 생성된 순차적 시간에 따라 산출물 객체의 순차 버전을 적용하였다. 그리고, 대체 버전(alternative version)은 특정 객체를 다른 플랫폼에 적용시킨 경우와 같이 새로운 목적을 가지고 생성된 버전이다.

산출물 객체 저장소는 특정 객체의 순차, 대체 버전에 대한 버전화와 버전 집합에 대한 자동 관리 기능을 가져야 한다.

3.1 종속 관계, 포함 관계

산출물 객체 저장소는 저장소에 저장된 데이타 요소들간의 다양한 관계들을 관리하는데, 이들 관계는 단순한 객체간의 의존적 관계와 객체간의 결합 관계 등이 있다. 이러한 개발 객체들간의 관계를 유지하는 것을 연결 관리라고도 부른다 [14].

그림 3에서 객체 B는 객체 C, D와 종속 관계에 있으며, 객체간의 종속 관계는 식 (1)과 같이 나타낼 수 있다.

객체 B가 객체 C, D와 종속 관계에 있다는 것은 C객체, D객체가 B객체에 종속되어 있는 것이며, B객체에 의해서 C객체, D객체는 영향을 받게 된다.

$$\textit{Dependency ID} = \{\textit{Source Object}\ ,\ \textit{Target Object}\} \quad (1)$$

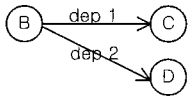


그림 3 산출물 객체간의 종속 관계

즉, 단일 산출물 객체 B는 단일 산출물 객체 D 와 Dependency 2라는 종속관계를 가지며, 식 (1)에 의해 Dependency 2 = { Object B, Object D }와 같이 나타낼 수 있다.

집단(aggregation) 산출물 객체는 여러 개의 산출물 객체의 콜렉션(collection)이다. 다음의 그림 4는 단일 산출물 객체간의 포함(containment) 관계를 보이고 있다.

집단 산출물 객체는 하나 이상 여러 개의 산출물 객체로 구성된 새로운 산출물 객체라고 볼 수 있으며, 다음 식 (2)와 같이 나타낼 수 있다.

Aggregation ID={*Object* 1, ... , *Object N*} (2)

그림 4에서 Aggregation 2는 객체 D, 객체 E로 구성되는 데, Aggregation 2 = {Object D, Object E}로 나타낼 수 있다.

예를 들어, 어떤 작업을 수행하는 집단 산출물 객체 A의 구성이 그림 5와 같다고 할 때, 버튼 객체 B, C와 텍스트 박스 객체 D는 집단 객체 A에 포함된 단일 객체이고, 그들 간의 종속 관계는 그림 6(b)와 같다.

그림 5의 예제는 그림 6(a)에서와 같이 패널 객체 A와 객체 B, C, D는 포함 관계에 있으며, 버튼 객체 B와 C가 소스가 되고, 텍스트박스 객체 D가 타겟이 되는 종속 관계를 가진다. 이 예제의 경우에 객체 B와 객체 D가 종속 관계를 가진다는 것은 버튼 객체 B가 실행되었을 경우의 결과 값이 텍스트박스 객체 D에 나타남을 의미한다.

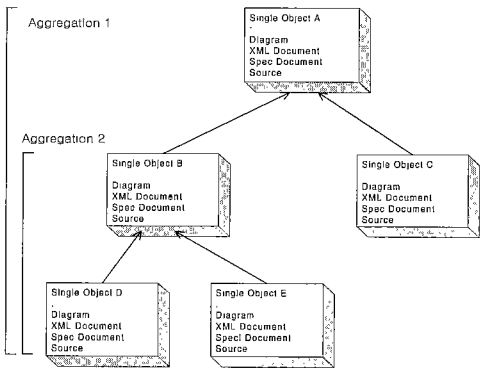


그림 4 단일 산출물 객체와 집단 산출물 객체

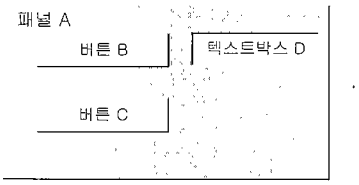


그림 5 산출물 객체 예제

(a) 포함 관계 (b) 종속 관계

그림 6 예제 산출물 객체간의 관계

3.2 버전 관계

산출물 객체의 버전 정보는 동일 플랫폼에서 시간의 순서에 따른 버전 관계와 특정 객체에 대해 다른 플랫폼을 목표로 개발되는 대체 버전 관계에 대한 정보를 가진다. 또한, 특정 루트 버전에 대해 대체 버전 관계에 있는 객체 중에서 같은 플랫폼을 가진 객체는 별도로 표시해두어 검색할 수가 있다.

산출물 객체의 버전은 그림 7과 같이 시간의 순서에 따른 순차 버전 관계와 특정 목적에 따라 대안적으로 객체가 개발되는 대체 버전 관계로 구분할 수 있다.

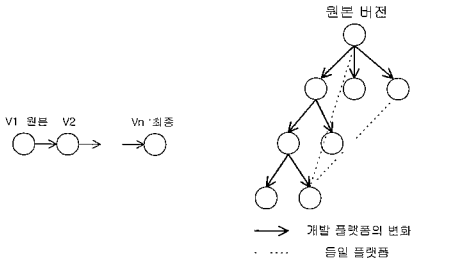
① 시간적 추이에 의한 버전 (linear version)

LV = { V1, V2,... ,Vn}

② 개발 플랫폼 변경에 의한 버전 (alternative version)

AV = (predecessor version, {successor version})

그림 7(a)에서와 같이 순차 버전은 특정 객체가 같은 플랫폼에서 시간적인 순서에 따라 버전화되는 버전이다. 순차 버전에서 선조 버전은 시간적으로 현재 버전에 대해 앞서는 버전이며, 후손 버전은 현재 버전의 다음 버전이다.

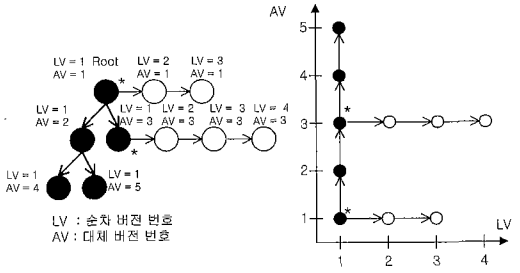


(a) 객체의 순차 버전 (b) 객체의 대체 버전

그림 7 객체의 버전화

대체 버전은 특정 객체가 개발 플랫폼 변경으로 인해 발생하는 버전이며, 그림 7(b)에서와 같이 굵은선으로 표시한다. 그리고, 점선으로 연결된 부분은 대체 버전들 중에서 서로 플랫폼이 같은 버전을 연결한 것이다.

산출물 객체의 전체 버전은 V = (LV, AV)로 표현될 수 있다. 본 논문에서 대체 버전은 객체의 버전을 결정하는 여러 요소 중에서 플랫폼을 변화 요소로 하는 버전이다. 즉, 특정 산출물 객체에 대한 대체 버전은 그 산출물의 구현 환경이 다르다는 것을 의미한다. 그러나, 산출물의 구현 형태가 마이크로소프트의 COM과 같이 플랫폼 독립적일 경우에는 순차적 버전만 적용될 수도 있다.



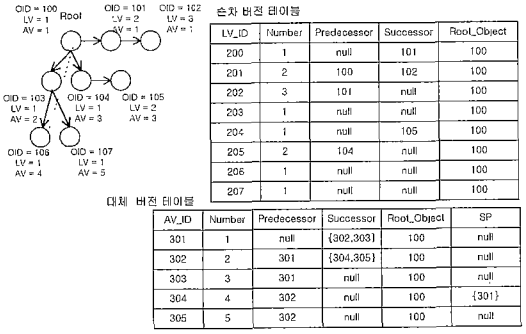
(a) 버전 그래프 (b) 2차원 좌표상의 버전

그림 8 순차 버전과 대체 버전

버전화된 객체의 버전 관계는 그림 8(a)에서와 같이 순차 버전과 대체 버전의 그래프 형태로 표현할 수 있으며, 이 버전 그래프는 그림 8(b)와 같이 순차 버전과 대체 버전의 병렬 관계로 표현될 수 있다. x축을 따라 변화하는 순차 버전은 특정 플랫폼에서 시간에 따라 산출물 객체가 타임 스탬프되는 과정을 보여 주고 있으며, y축을 따라 변화하는 대체 버전은 플랫폼에 따라 산출물 객체가 변화되는 과정을 보여준다. 대체 버전을 결정하는 변화 요소에는 여러 가지가 있는 데, 본 논문에서는 그 변화요소를 플랫폼으로 한정해서 산출물 객체의 버전 관계를 2차원 상에 표현할 수 있었다. 만약, k개의 변화요소를 고려할 경우 그 버전은 k차원 상에 표현되어야 할 것이다.

버전화된 객체의 순차 버전과 대체 버전은 관련된 데이타베이스의 버전 테이블에 한 튜플로 매핑 될 수 있으며, 객체와 특정 버전은 OID와 VID의 유일한 식별자에 의해 구별될 수 있다. 또한, 버전화된 객체들은 그 버전 객체들의 루트 객체에 대해 리스트의 형태로 테이블에 저장되어 유지된다.

그림 9에서와 같이 순차 버전과 대체 버전은 각각의



순차 버전 테이블

LV_ID	Number	Predecessor	Successor	Root_Object
200	1	null	101	100
201	2	100	102	100
202	3	101	null	100
203	1	null	null	100
204	1	null	105	100
205	2	104	null	100
206	1	null	null	100
207	1	null	null	100

대체 버전 테이블

AV_ID	Number	Predecessor	Successor	Root_Object	SP
301	1	null	{302,303}	100	null
302	2	301	{304,305}	100	null
303	3	301	null	100	null
304	4	302	null	100	{301}
305	5	302	null	100	null

그림 9 산출물 객체 버전의 테이블 매핑

테이블에 매핑된다. 순차 버전은 같은 원본 객체에 대해 동일한 플랫폼에서 개발되는 객체들에 대한 버전이며, 순차 버전 테이블에는 순차적으로 발생하는 특정 객체의 순차 버전 번호, 선조 버전, 후손 버전에 대한 정보를 테이블의 한 튜플에 매핑시킨다. 대체 버전은 특정 객체의 개발 플랫폼 변화에 의해 발생하는 버전이며 대체 버전 테이블에는 원본 객체나 이전 객체의 대체 버전에 대한 대체 버전 번호, 선조 버전, 후손 버전에 대한 정보를 테이블의 한 튜플에 매핑시킨다. 대체 버전 테이블의 'SP' 칼럼은 대체 버전 객체 중에서 플랫폼이 같은 버전에 대한 대체 버전 식별자의 집합이다.

4. 내용 기반 검색

저장소에 저장된 산출물 객체 중에서 UML 다이아그램 형태로 표현된 클래스 다이아그램이나 시퀀스 다이아그램 자체의 정보를 저장, 검색 및 공유하기 위해서는 그래픽 형태의 다이아그램을 단순 텍스트 형태의 문서나 구조 기반의 문서 형태로 변환하여야 한다.

본 논문에서는 UML 모델의 클래스 다이아그램을 산출물 객체의 정적인 정의를 표현하는 다이아그램으로, 시퀀스 다이아그램을 산출물 객체의 동적인 정의를 표현하는 다이아그램으로 간주한다. 이 UML 모델의 정적인 클래스 다이아그램과 동적인 시퀀스 다이아그램을 UML 메타모델의 구성요소를 이용해 XML 문서 형태로 표현한다.

UML 모델의 클래스 다이아그램과 시퀀스 다이아그램을 XML 문서화 하면, 기존의 DBMS에 저장하여 내용 기반 검색을 할 수가 있으며, 웹 상에서도 XQL, XSL, DOM 등을 이용해 검색 및 디스플레이 할 수 있다. 또한, XML 문서로 변환된 산출물 객체는 상이한 플랫폼간의 데이타 정보 교환을 용이하게 할 수 있다.

4.1 UML 모델의 XML 문서화

산출물의 내용 기반 검색과 공유를 위한 UML 모델의

XML 문서화 작업은 UML 모델을 표현하는 UML 메타 모델의 구성 요소 중에서 UML 모델의 개념(메타 클래스)를 나타내는 추상 구문(abstract syntax)을 이용한다. UML 메타 모델의 추상 구문은 클래스 다이아그램 형태로 구성되어 있으며 UML 모델의 구성 요소와 구성 요소간의 연관 관계를 나타낸다.

4.1.1 클래스 다이아그램의 표현

클래스 다이아그램은 객체 타입과 그들 사이에 존재하는 여러 가지 정적인 관계를 표현하며, 클래스 다이아그램의 기본적인 구성 요소는 그림 10과 같다.

그림 10의 요소들은 클래스의 애트리뷰트와 그 클래스가 가진 연산을 정의하고, 관련된 클래스간의 항해성(navigability), 카디날리티를 나타낸다. 그리고, 부품 관계(part-of)를 나타내는 집단(aggregation) 연관성과 한 부품 객체가 오직 하나의 전체에 속해 있고 부품의 생명주기가 전체와 같이 하는 복합(composition) 연관성도 표현하고 있다.

그림 10에서 클래스의 기본 구성 요소를 UML 메타 모델내의 Foundation 패키지의 추상 구문을 기반으로 XML DTD 형태로 표현한 결과는 그림 11과 같다.

그림 12는 클래스 다이아그램에서 클래스 간의 일반화 관계, 연관 관계, 종속 관계, 인스턴스 객체, 연관 클래스를 표현하는 요소를 나타내며, XML DTD는 그림 13과 같다.

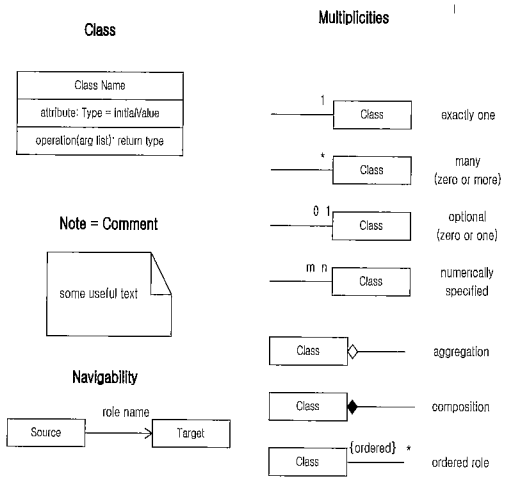


그림 10 클래스 다이아그램의 기본 구성 요소

```
<!ENTITY Classifier "(Class | AssociationClass)">
<!ENTITY %ClassifierFeature "(Attribute | Operation | Method)*">
<!ELEMENT Class (Note*, Name, IsActive?, ClassifierFeature;)>
<!ELEMENT IsActive ("true" | "false")>
<!ELEMENT Attribute (Name, DataType?, Visibility?, initialValue?,
TargetScope?, Changeable?, Multiplicity?)>
<!ELEMENT Visibility ("public" | "protected" | "private")>
<!ELEMENT InitialValue (#PCDATA)>
<!ELEMENT TargetScope ("instance" | "classifier")>
<!ELEMENT Changeable ("none" | "frozen" | ""addonly")>
<!ELEMENT Multiplicity ("1" | "*" | "0..1" | "m. n")>
<!ELEMENT Operation (Name, Visibility?, ReturnType?,
Concurrency?, IsAbstract?, IsLeaf?, IsRoot?, OwnerScope?,
Parameter*)>
<!ELEMENT Parameter (Name, Type?, DefaultValue?, Kind?)>
<!ELEMENT ReturnType (#PCDATA)>
<!ELEMENT Concurrency ("seuential" | "guarded" | "concurrent")>
<!ELEMENT IsAbstract ("true" | "false")>
<!ELEMENT IsLeaf ("true" | "false")>
<!ELEMENT IsRoot ("true" | "false")>
<!ELEMENT OwnerScope ("instance" | "classifier")>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT DefaultValue (#PCDATA)>
<!ELEMENT Kind ("in" | "out" | "inout" | "return")>
<!ELEMENT AssociationClass (Name, AttachedAssociation
,%ClassifierFeature;)>
<!ELEMENT AttachedAssociation (#PCDATA)>
```

그림 11 클래스 기본 구성 요소에 대한 XML DTD

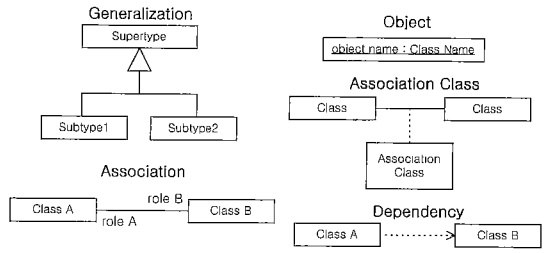


그림 12 클래스 다이아그램의 기타 구성 요소

```
<!ELEMENT ClassDiagram (Name, (Note
| %ExportedCoreElements;)*)>
<!ENTITY %ExportedCoreElements "(Class | Dependency
|Generalization |Association |AssociationClass | Object )*">
<!ELEMENT Generalization (Discriminator, Parent, Child)>
<!ELEMENT Discriminator (#PCDATA)>
<!ELEMENT Parent (#PCDATA)>
<!ELEMENT Child (#PCDATA)>
<!ELEMENT Association (AssociationName*, AssociationEnd>
<!ELEMENT AssociationName (Name, Direction?)>
<!ELEMENT Direction (Source, Target)>
<!ELEMENT Source (DataType, RoleName?)>
<!ELEMENT Target (DataType, RoleName?)>
<!ELEMENT RoleName (#PCDATA)>
<!ELEMENT AssociationEnd (DataType, Name?, AggregationKind?,
Multiplicity? )>
<!ELEMENT AggregationKind ("AggregationWhole" |
AggregationPart" | "CompostionWhole")>
<!ELEMENT Dependency (Note*, Name?, Supplier, Client)>
<!ELEMENT Client (#PCDATA)>
<!ELEMENT Supplier (#PCDATA)>
<!ELEMENT Object (Name, Classifier)>
```

그림 13 클래스 다이아그램의 기타 구성 요소에 대한 XML DTD

4.1.2 시퀀스 다이아그램의 표현

시퀀스 다이아그램은 특정 도메인에 대한 객체지향 모델에서, 각각의 객체간에 발생하는 메시지의 흐름을 보여주는 것으로, 기본적인 구성은 그림 14와 같다. 사각형 박스는 객체를 나타내고, 수직선은 그 객체의 생명

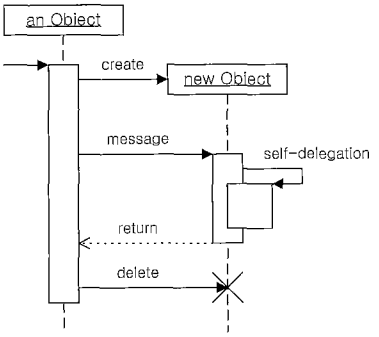


그림 14 시퀀스 다이아그램의 구성 요소

```
<!ELEMENT SequenceDiagram (Name, (Note
| %ExportedCoreElements; | Collaboration)*)>
<!ENTITY %ExportedCoreElements "(Class | AssociationClass |
Object )*">
<!ELEMENT ActionSequence (Name?, Action*)>
<!ELEMENT Action (Name?, ActionType?, Target?, Argument?,
Request?)>
<!ELEMENT ActionType ("CreationAction" | "CallAction" |
"LocalInvocation" | "ReturnAction" | "SendAction" |
"DestroyAction")>
<!ELEMENT Target (#PCDATA)>
<!ELEMENT Argument (#PCDATA)>
<!ELEMENT Request ("operation | "signal")>
<!ELEMENT Collaboration (Name, Message*)>
<!ELEMENT Message (Sender, Receiver, Activator?, Predecessor?,
        GuardCondition?, SequenceExpression, ReturnValue?,
        Signature?)>
<!ELEMENT Sender (#PCDATA)>
<!ELEMENT Receiver (#PCDATA)>
<!ELEMENT Activator (#PCDATA)>
<!ELEMENT Predecessor (#PCDATA)>
<!ELEMENT SequenceExpression (#PCDATA)>
<!ELEMENT ReturnValue (#PCDATA)>
<!ELEMENT Signature (#PCDATA)>
```

그림 15 시퀀스 다이아그램의 XML DTD

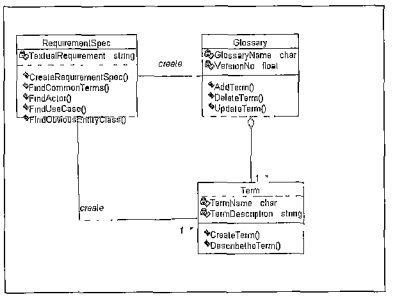


그림 16 요구사항분석 UML 클래스 다이아그램

선(life line)을 나타낸다. 그리고, 화살표를 가진 실선과 점선이 객체간에 주고받는 메시지이며, 실선은 객체의 생성, 메시지 전달 등을 나타내고, 점선은 요청한 메시지에 대한 리턴 값을 의미한다.

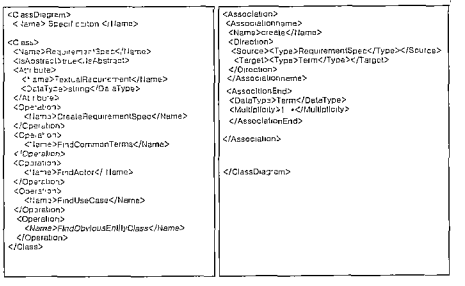
시퀀스 다이아그램의 구성 요소를 UML 메타 모델의 Behavioral 패키지의 추상 구문을 기반으로 XML DTD로 표현하면 그림 15와 같다.

4.2 UML 모델의 XML 문서화 예

그림 16은 '요구사항분석' 이라는 UML 클래스 다이아그램의 예이다.

이 클래스 다이아그램을 UML 메타 모델의 XML DTD를 이용해 XML 문서화 한 결과는 그림 17과 같다.

그림 17(a)는 그림 16의 클래스 다이아그램에 속한 클래스를 정의하며, 그림 17(b)는 그 클래스 인스턴스들 사이의 연관 관계를 표현한 것이다.



(a) 클래스와 Operation (b) 연관

그림 17 요구사항분석 클래스 다이아그램의 XML 문서

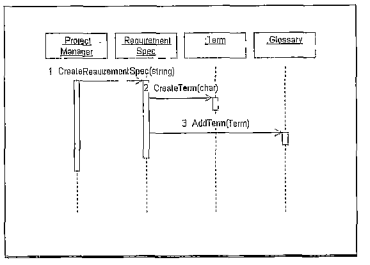


그림 18 요구사항분석 시퀀스 다이아그램

```
<SequenceDiagram>
  <Name>Requirement_Capture</Name>
    <Object>
      <Name>ProjectManager</Name>
    </Objet>
...
<ActionSequence>
  <Action>
    <Name>CreateRequirementSpec</Name>
    <ActionType> CallAction </ActionType>
  </Action>
  <Action>
    <Name>CreateTerm</Name>
    <ActionType> CallAction </ActionType>
  </Action>
  <Action>
    <Name>AddTerm</Name>
    <ActionType> CallAction </ActionType>
  </Action>
</ActionSequence>
<Collaboration>
<Name>CreateRequirementSpec</Name>
<Message>
  <Sender>ProjectManager</Sender>
  <Receiver>RequirementSpec</Receiver>
  <SequenceExpression>1</SequenceExpression>
 <Signature>CreateRequirementSpec(string)</Signature>
</Message>
</Collaboration>
..
</SequenceDiagram>
```

그림 19 요구사항 분석 시퀀스 다이아그램의 XML 문서

그림 18은 '요구사항분석' 클래스 다이아그램에 대한 객체간의 동적 행위를 표현하는 시퀀스 다이아그램의 예이다

그림 18의 시퀀스 다이아그램에 대한 XML 문서는 그림 19와 같다.

5. 관계 및 내용 기반 검색 시스템의 구현

5.1 산출물의 메타 스키마

산출물 객체 저장소의 저장 및 검색을 위한 메타 스키마는 그림 20과 같다. 한 개발자는 여러 개의 산출물 객체를 개발할 수 있으며, 한 플랫폼은 여러 산출물 객체를 가질 수 있다. 산출물 객체는 시간적 변화에 따른 순차 버전과 플랫폼의 변화에 따른 대체 버전을 가질 수 있다.

그리고, 산출물 객체는 자신을 포함한 다른 산출물 객체들과 종속 관계를 가질 수 있으며, 집단(aggregation) 형 산출물 객체의 관련 객체 집합 정보는 Item_Set 이라는 리스트 타입의 필드에 저장한다.

산출물 객체 ID와 버전 ID는 유일한 산출물 객체와 버전 식별자 값을 가져야 하므로 산출물 객체 테이블과 버전 테이블에 SERIAL 타입의 ID 필드를 추가하였다.

또한, 테이블간의 참조 시에 사용하기 위해 개발자 테이블의 주민등록번호(SSN)과 같이 유일한 식별자가 있

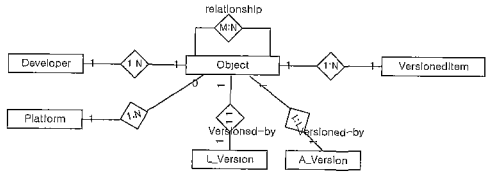


그림 20 저장소의 메타 스키마에 대한 E-R 다이아그램

Column Name	Type	Con.	Description
Object_ID	SERIAL	PK	객체 식별자
Name	VARCHAR(30)		객체 이름
Date	DATETIME		객체 개발 일자
XML_Document	XML		XML 문서
Spec_Document	DOCUMENT		DOC 형태의 MS WORD 문서
Class_Diagram	GIF		클래스 다이아그램 파일
Sequence_Diagram	GIF		시퀀스 다이아그램 파일
Source_Code	SOURCE		객체 구현 코드
L_Version	INTEGER	FK	순차 버전의 번호
A_Version	INTEGER	FK	대체 버전의 번호
Developer	INTEGER	FK	개발자 ID
Platform	INTEGER	FK	플랫폼 ID
Item_Set	SET of object		집단 객체 ID의 집합
Aggregation	BOOLEAN		집단 객체인가?

그림 21 객체 테이블

Column Name	Type	Con.	Description
SSN	VARCHAR(14)	PK	주민등록번호
Name	VARCHAR(30)		개발자 이름
Company	VARCHAR(15)		개발자 회사
e-mail	VARCHAR(20)		e-mail 주소

그림 22 개발자 테이블

Column Name	Type	Con.	Description
Platform_ID	SERIAL	PK	플랫폼 식별자
Machine_Name	VARCHAR(20)		컴퓨터의 기종
OS_Name	VARCHAR(20)		운영체제 종류 및 버전
Lang_Name	VARCHAR(20)		개발 언어 종류
Window	VARCHAR(20)		윈도우 종류
DBMS	VARCHAR(20)		DBMS의 종류

그림 23 플랫폼 테이블

을 경우를 제외한 모든 테이블에 테이블 식별자인 ID 필드를 추가하였으며, 테이블간의 ID구별은 (테이블.ID)의 형태로 구분한다.

그림 21, 22, 23은 산출물 객체 저장소에 대한 메타 스키마중에서 객체와 개발자, 플랫폼 정보를 객체관계 DBMS인 Informix Universal Server의 테이블에매핑한 것이다.

5.2 산출물 관리 모듈

저장소내에 저장된 산출물 데이타에 대해 산출물 저장소의 DBMS 자체 기능과 기타 부가적인 여러가지 기능을 얻기 위해 산출물 데이타의 기본적인 저장 방식을 정의하는 메타 스키마외의 산출물 데이타 관리 모듈이 필요하다.

5.2.1 산출물 객체간의 종속 관계 관리

본 논문에서는 저장소에 저장되는 대상 산출물들의 단일 객체간의 종속(dependency) 관계 관리를 위해 그림 24와 같은 관계 테이블을 구성하였다.

산출물 객체의 삭제 시 관계 정보는 SQL 구문의 CASCADE DELETE ON 절을 이용하여, 관련 산출물

Column Name	Type	Con.	Description
Relationship_ID	SERIAL	PK	관계 식별자
Name	VARCHAR(30)		컴포넌트 관계명
Source_ID	INTEGER	FK	관계의 Source (Object ID)
Target_ID	INTEGER	FK	관계의 Target (Object ID)

그림 24 종속 관계 정보 관리를 위한 관계 테이블

객체의 삭제 시 관계 정보가 자동으로 삭제 되도록 한다. 집단(aggregation)형 산출물 객체의 관련 산출물 객체의 리스트는 산출물 객체 테이블의 리스트 타입 필드에 유지된다.

5.2.2 산출물 객체간의 버전 관리

본 논문에서는 산출물 객체의 버전 관계의 표현과 버전 정보의 관리를 위해 그림 25, 그림 26과 같은 버전 테이블과 특정 버전 객체의 순차 버전과 대체 버전에 대한 버전 리스트 관리를 위해 그림 27과 같은 버전 항목 테이블을 구성하였다.

Column Name	Type	Con.	Description
LV_ID	SERIAL	PK	Linear Version 식별자
Number	INTEGER		버전 번호
Predecessor	INTEGER		현재 버전의 상위 순차 버전의 객체 ID
Successor	INTEGER		현재 버전의 하위 순차 버전의 객체 ID
Root	BOOLEAN		Root 버전인가에 대한 논리 값 (default='T')
Final	BOOLEAN		Final 버전인가에 대한 논리 값 (default='F').
Root_Object	INTEGER	FK	해당 버전 객체의 Root 객체 ID

그림 25 순차 버전 테이블

Column Name	Type	Con.	Description
AV_ID	SERIAL	PK	대체 버전 식별자
Number	INTEGER		버전 번호
Predecessor	INTEGER		현재 버전의 상위 대체 버전의 객체 ID
Successor	SET of object		현재 버전의 하위 대체 버전의 객체 ID
Root_Object	INTEGER	FK	해당 버전 객체의 Root 객체 ID
SP	SET of object		플랫폼이 같은 대체 버전의객체 ID 집합

그림 26 대체 버전 테이블

Column Name	Type	Con.	Description
Vlist_ID	SERIAL	PK	버전화 된 객체의 항목에 대한 식별자
Item_List	LIST		버전화 된 객체 리스트
Version_Type	VARCHAR(2)		'AV' or 'LV'
Root_Object	INTEGER	FK	Root 객체 ID

그림 27 버전의 리스트 관리를 위한 버전 항목 테이블

새로운 산출물 객체의 삽입 시에 버전을 생성하고, 버전 리스트 테이블에 버전화된 객체를 삽입하기 위한 일련의 작업들을 의사코드 형태로 나타내면 그림 28과 같다. 그림 28의 2행에서 5행까지는 의사코드의 시작부분이며 의사코드에 필요한 변수를 설정한다. 6행은 산출물 객체의 개발이 종료된 객체이다.

```
1:  procedure Add_Version(io)
2:  // io : 새로 입력된 객체
3:    l_version  새로운 객체의 순차 버전
4     a_version : 새로운 객체의 대체 버전
5.    vitem : 버전화된 객체의 리스트 //
6.  if io(final) =1 then  개발이 종료된 버전 객체 이다
7.  if io(root) = 1   then  // 새로 입력된 객체가 루트이면 //
8:  l_version(number) ← 1      // 순차 버전 번호에 1을 삽입 //
9   a_version(number) ← 1      // 대체 버전 번호에 1을 삽입 //
10: vitem(root 객체) ← 현재 객체 ID
11: vitem(version_type) ← 'LV'      // 버전 타입을 순차 버전으로 입력 //
12: vitem(list) ← l_version(ID)     // 순차 버전 리스트 생성 //
13: vitem(root 객체) ← 현재 객체 ID
14: vitem(version_type) ← 'AV'      // 버전 타입을 대체 버전으로 입력 //
15: vitem(list) ← a_version(ID)     // 대체 버전 리스트 생성 //
16: io(l_version) ← 현재 순차 버전 ID
17  io(a_version) ← 현재 대체 버전 ID
18. else if io(root) = 0 and io(platform) = 이전 버전 객체(platform) then
19  l_version(number) ← 이전 버전 객체의 l_version(number) + 1
20: a_version(number) ← 이전 버전 객체의 a_version(number)
21: 현재 객체(root_object) = vitem(root_object) and
22: vitem(version_type)='LV'인 vitem(list)중에서 이전 버전 객체를 포함하는
    vitem(list) ← 현재 객체의 version(ID)   // 순차 버전 리스트에 현재 객체 추가 //
23: io(l_version) ← 현재 순차 버전 ID
24: io(a_version) ← 이전 버전 객체의 대체 버전 ID
25: else if io(root) = 0 and io(platform) ≠ 이전 버전 객체(platform) then
26: l_version(number) ← 1
27: n ← 현재 객체(root_object) = a_version(root_object)인 대체 버전 번호의 최대
    값 a_version(number) ← n + 1
28  현재 객체(root_object) = vitem(root_object) and vitem(version_type)='AV'인
    vitem(list) ← 현재 객체의 a_version(ID)   // 대체 버전 리스트에 현재 객체 추가 //
29: vitem(root 객체) ← 현재 버전 객체의 root 객체 ID
30: vitem(version_type) ← 'LV'
31  vitem(list) ← l_version(ID)     // 순차 버전 리스트 생성 //
32. io(l_version) ← 현재 순차 버전 ID
33. io(a_version) ← 현재 대체 버전 ID
34. end Add_version(io)
```

그림 28 산출물 객체의 버전화 의사코드

7행에서 17행까지는 산출물 객체가 루트(root) 객체인 경우이며, 순차 버전과 대체 버전은 모두 1이 되고 해당 루트 객체에 대해 순차 버전 리스트와대체버전 리스트를 생성한다.

18행에서 24행까지는 산출물 객체가 루트나 최종(final) 객체가 아니고, 이전 객체와 플랫폼이 같은 경우이며 순차 버전화 되는 과정이다. 순차 버전화 될 경우 20행에서와 같이 대체 버전은 이전 버전과 동일하며 순차 버전은 19행과 같이 이전 객체에서 1 증가한다. 22행은 현재 객체를 해당 순차 버전 리스트에 삽입하는 행이다. 25행에서 33행까지는 이전 객체와 플랫폼이 다른 경우이며 대체 버전화 되는 과정이다. 대체 버전화는 27행과 같이해당 루트 객체에 대한 대체 버전을 1증가시키고 현재 객체를 28행에서 대체 버전 리스트에 추가한다. 29행에서 31행은 해당 객체에 대해 순차 버전 리스트를 새로이 생성하는 과정이다.

5.3 내부 저장 구조 및 시스템 구조

본 논문에서 제안한 UML 산출물의 관계 및 내용 기반 검색을 위한 시스템은 멀티미디어 산출물 객체의

메타 정보 처리를 위한 부분과 UML 다이아그램에 대한 XML 문서 처리 부분으로 구성된다.

5.3.1 내부 저장 구조

산출물 객체 저장소의 객체 데이타에 대한 산출물의 저장 과정은 다음과 같다. UML을 기반으로한 객체지향 개발 단계의 멀티미디어 산출물 중 클래스 다이아그램과 시퀀스 다이아그램은 XML 문서 형태로 변환된다. 이 *.XML 문서는 CLOB 형태로 객체관계 DBMS에 저장되고, 그 중에서 XML 문서는 eXcelon에도 저장되어 처리된다. 그리고 클래스 다이아그램과 시퀀스 다이아그램의 이미지도 BLOB 형태로 객체관계 DBMS에 저장된다.

산출물 객체의 메뉴얼 문서는 워드 문서 자체가 CLOB 형태로 저장되고, 구현코드도 BLOB 형태로 코드 자체가 저장된다.

그리고, 사용자가 입력하는 메타 스키마 정보와 산출물 간의 관계 정보도 객체관계 DBMS의 해당 테이블의 필드에 저장된다. 버전 기능은 객체관계 DBMS 프로시저의 실행에 의해 처리된다.

5.3.2 시스템 구조

UML을 기반으로 한 객체지향 개발 단계의 멀티미디어 산출물 객체 관리를 위한 객체 저장소 시스템은 그림 29와 같이 구성되며, 이 시스템에서 사용자는 클라이언트 쪽의 웹 브라우저를 이용해 산출물 객체 저장소에 대화식 사용자 인터페이스 방식으로 저장 및 검색 작업을 요청한다. 사용자가 입력한 산출물 객체의 메타 정보와 산출물 객체의 버전화에 대한 처리는 Informix Universal Server에서 Web DataBlade Module과

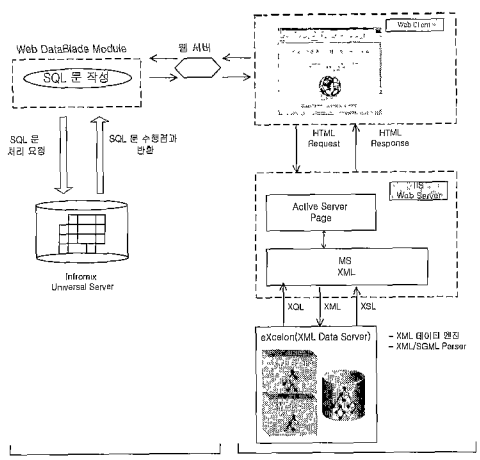


그림 29 시스템 구성도

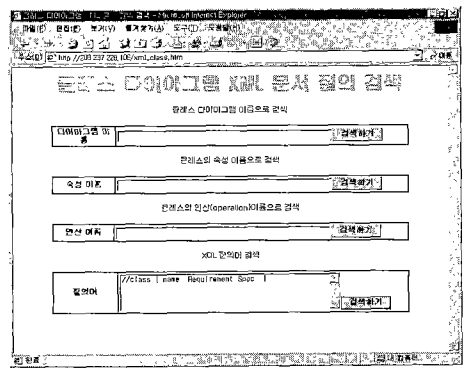


그림 30 산출물 객체의 메타 정보를 이용한 검색

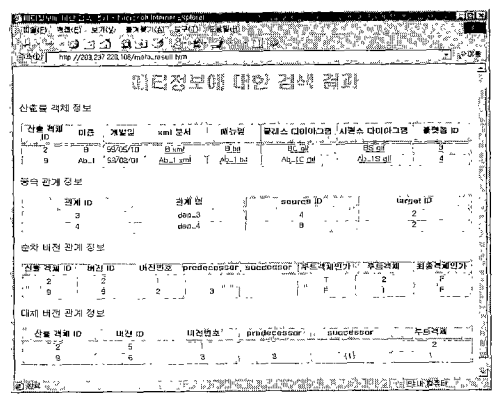


그림 31 개발자 이름에 의한 산출물 객체 검색 결과

Webdriver를 통해 처리된다. 현재 IUS에서는 XML 문서의 저장 및 검색을 지원하지 않으므로 XML 문서 형태로 변환된 산출물 객체의 저장 및 검색 작업은 eXcelon XML 데이타 서버에 의해서 수행된다.

산출물 객체에 대한 검색 작업의 결과 값은 IUS와 eXcelon의 웹 서버를 통해 HTML 형태로 반환된다.

5.3.3 메타 정보를 이용한 산출물 객체 관계 정보 검색

그림 30은 산출물 객체의 메타 정보 중에서 개발자 이름 = 'kim'인 개발자가 개발한 산출물 객체를 검색하는 화면이고 그 결과는 그림 31과 같다.

그림 31과 같은 객체의 관계 정보는 산출물 객체 간의 종속, 포함 관계, 버전 관계를 이용한 산출물 객체 정보 추출에 활용될 수 있다.

5.3.4 XML 문서에 대한 내용 기반 검색

그림 32는 XML 문서 형태로 변환된 클래스 다이아그램의 문서 중에서 <class>의 <name>이 'Requirement Spec'인 문서를 XQL 질의문 형식으로 검색하는 화면이다.

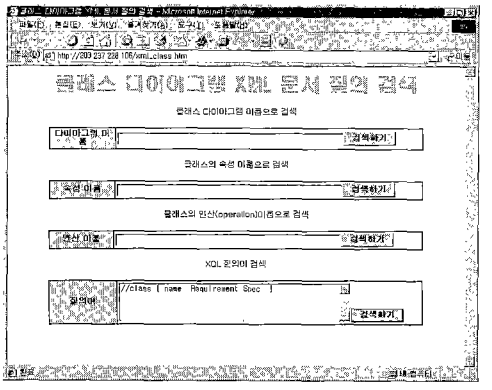


그림 32 XML 문서에 대한 질의

6. 결 론

본 논문에서는 UML을 기반으로 한 객체지향 개발에서 발생하는 멀티미디어 산출물 객체를 저장, 관리가 가능한 저장소를 설계하고, 저장소 내의 산출물 객체 데이타에 대한 관계 및 내용기반 검색을 위한 산출물 관리 기법을 제시하고, 상용 DBMS을 이용하여 산출물 관리 모듈을 구현하였다. 또한, 본 논문에서 제시한 산출물 관리 기법을 이용하여 인터넷 기반으로 멀티미디어 자료처리, 산출물 객체간의 관계 관리, 버전 관리, XML 문서처리가 가능한 시스템을 구축하였다.

산출물 객체 저장소의 요구사항을 분석하기 위하여 대상 산출물 객체를 구성 형태, 구성 내용에 따라 분류하고, 멀티미디어 산출물 객체 저장소의 기능을 분석하였다. 그리고, 객체 저장소의 저장 대상 산출물 객체의 기반이 되는 객체지향 분석/설계를 위한 표준 기술인 UML 모델에 대해 분석하였다.

산출물 객체를 구성 형태나 내용에 따라 관리하고 산출물 객체간의 종속 관계, 버전 관계의 관리를 위한 관리 기법을 제시하였다.

산출물 객체 저장소의 관계 관리 모듈 구현을 위해 관계 관리 테이블과 버전 관리 테이블, 버전 리스트 테이블을 구성하고 관련 프로시저를 작성하였다.

이 관계 테이블, 버전 테이블, 버전 리스트 테이블내에 저장된 객체간의 종속, 버전 관계 정보를 통해 관련 객체의 관계 기반 검색이 가능하다. 그리고, UML을 기반으로 한 객체지향 분석/설계 단계의 산출물 객체를 서로 다른 환경에서 공유 또는 교환하기 위해 UML 메타 모델의 추상 구문을 기반으로 UML 다이아그램 산출물을 XML 문서 형태로 변환하였다. 이 다이아그램에 대한 XML 문서는 산출물 객체 저장소와 관련된 다양한 도구들 간의 산출물 객체의 전달을 위한 수단과 저장 산출물 객체에 대한 내용 기반 검색의 수단으로 활용될 수 있다.

추후 연구 과제로는 UML 다이아그램에서 XML 문서로의 변환 과정에서 UML 모델에 대한 유효성(validation)을 검사해줄 수 있는 방법이 추가되어야 할 것이다. 또한 분산객체 환경의 지원을 위한 분산 객체 기술인 CORBA와 XML과의 상호연동을 고려하여야 할 것이다.

본 논문에서는 산출물 객체 저장소의 객체 데이타의 저장을 위해 객체 관계 DBMS를 사용하였는데, 추후에는 산출물 객체의 테이블 매핑이 필요없고 검색시 조인 연산이 없는 객체지향 DBMS에 적용해보고, 제시된 객체 저장소의 기능 검증을 위해 특정 응용에 대한 적용 및 기능 개선 작업이 수행되어야 할 것이다.

참 고 문 헌

[1] Martin Fowler, *UMLbDistilled,* Addison-Wesley.
[2] Jacobson, I., et.al., *Object-Oriented Software Engineering*, 1992, Addison-Wesley.
[3] Rumbaugh,J.,et.al., *Object-Oriented Modeling and Design*, 1991, Prentice-Hall.
[4] 나연묵, "산출물 객체의 효율적 관리를 위한객체 저장소설계", 단국대학교 논문집 제33집 1998, pp.291-301.
[5] World Wide Web Consortium, XML Language Specification and related documents, at http://www.w3.org/TR/REC-xml.
[6] Jose A. Blakeley, "Data Access for the Masses through OLEDB," SIGMOD, 1996.
[7] J. Shanmugasundaram et.al. "Relational databases for querying XML documents: Limitations and opportunities," in Proc. of VLDB,Edinburgh, Scotland, 1999.
[8] Rational Software et.al., UML Semantics, OMG document number: ad/97-08-04.
[9] Rational Software et.al., UML Notation Guide, OMG document number: ad/97-08-05.Rational Software et.al., Object Constraint Language, OMG document number: ad/97-08-05.
[10] Rational Software et.al., object Constraint Language, OMG document number : ad/97-08-05
[11] 전세길, 나연묵, "객체지향 개발 산출물의 관리를 위한 데이타베이스 스키마 정의 언어", 한국정보과학회 춘계학술발표 논문집, 26권,1호, 1999. 4, pp.42-44.
[12] Philip A. Bernstein and Umeshwar Dayal, "An overview of Repository Technology, "inproc. VLDB, 1994, pp.705-713.
[13] Phillip A Bernstein et.al., "The Microsoft Repository," in proc VLDB, 1997.2.

[14] Roger S. .Pressman,S*oftwarengineering*,McGraw-Hill, 1998
[15] Cattell, R.G.G., *The Object Database Standard*, Morgan Kaufmann, 1994.
[16] Philip A. Bernstein, "Repositories and Object Oriented Databases," SIGMOD Record, Vol. 27, No.1, March 1998.
[17] Microsoft Corp., Microsoft Repository web site, http://www.microsoft.com/repository.
[18] Setrag Khoshafian and Razmik Abnous, *Object Orientation*, Wiley, 1995.
[19] Informix Universal Server Guide to SQL(한글판),한국 인포믹스.
[20] R.G.G. Cattell, *Object Data Management*, Addison Wesley, 1994.
[21] Reidar Conradi and Richard Westfechtfel, "Version Models for Software ConfigurationManagement," ACM Computing Surveys, Vol 30, No2, June 1998.
[22] http://www.yy.cs.keio.ac.kp/~suzuki/project/uxf.

전 세 길
1998년 단국대학교 컴퓨터공학과 졸업 (공학사). 2000년 단국대학교 대학원 컴퓨터공학과 졸업 (공학석사). 2000년 ~ 현재 단국대학교 대학원 컴퓨터공학과 박사과정 재학중. 관심분야는 객체지향, 객체지향 데이타베이스, 멀티미디어 데이타베이스, 지식관리시스템.

나 연 목
1986년 서울대학교 컴퓨터공학과 졸업 (공학사). 1988년 서울대학교 대학원 컴퓨터공학과 졸업 (공학석사). 1993년 서울대학교 대학원 컴퓨터공학과 졸업 (공학박사). 1991년 미국 IBM T.J.Watson 연구소 객원연구원. 1994년 ~ 1999년 DASFAA Steering Committee Secretary. 1996년 ~ 현재 한국정보과학회 데이타베이스연구회 운영위원. 1997년 ~ 1999년 한국멀티미디어학회 논문지 편집위원. 1997년 ~ 현재 서울특별시 정보화추진위원회 위원. 현재 한국정보과학회 논문지 편집위원. 2001년 DASFAA Program Committee Member. 1993년 ~ 현재 단국대학교 공학부 컴퓨터공학 전공 부교수. 관심분야는 데이타베이스, 객체지향 데이타베이스, 데이타 모델링, 데이타베이스 설계, 멀티미디어 데이타베이스, 멀티미디어 정보 검색, 멀티미디어 시스템.