SGML 문서 관리 시스템의 설계 및 구현 *

Design and Implementation of SGML Document Management System

김 용 훈(Yong-Hun Kim)** 이 원 석(Won-Suk Lee)**
류 은 숙(Eun-Suk Ryu)** 이 규 철(Kyu-Chul Lee)**
이 상 기(Sang-Ki Lee)*** 김 현 기(Hyun-Ki Kim)***
이 혜 란(Hae-Ran Lee)*** 주 종 철(Zong-Chul Zhoo)***

목 차

초 록

다가오는 21세기는 고도화된 정보화 사회가 될 것이며, 이러한 정보화 사회에서 필수적인 요소로는 기하급수적으로 발생되는 전자 문서를 관리하는 것이라 할 수 있다. 특히, 디지털 도서관(Digital Library), 인트라넷 상에서의 CSCW(Computer-Supported Cooperative Work), CALS(Commerce At the Light Speed) 등의 최근의 응용에서는 대량의 전자 문서를 효율적으로 저장하고, 관리하는 시스템이 요구되고 있다. 다양한 전자 문서의 형태 중 ISO에서 문서 교환 표준으로 제정한 SGML은 구조 정보로 사용할 수 있는 마크업(Markup)을 포함하고 있어, 문서를 구조 단위로 표현할 수 있다. 이는 대량의 전자 문서를 저장하거나 관리하는데 적합하다. 본 논문에서는 SGML 문서를 효율적으로 관리하기 위한 SGML 문서의 데이터 모델을 제시하고, 다양한 구조의 SGML 문서에 대한 스키마 동적 생성 기능과 SGML 인스턴스 저장 기능 그리고 저장된 SGML 인스턴스를 문서 혹은 문서의 구조 단위로 꺼내오는 기능을 제공하는 SGML 문서 관리 시스템을 설계하고 구현하였다.

ABSTRACT

The 21st century will be the advanced Information society era. The management of very large quantity of electronic documents is important because new applications such as Digital Libraries, CSCW (Computer-Supported Cooperative Work) in Intranet, CALS(Commerce At the Light Speed) are emerging, which require the functionalities of efficient storing, searching and managing a bulk of electronic documents. SGML(Standard Generalized Markup Language) is an ISO Standard for representing structure information of electronic documents. This paper proposes an effective data model for storing and managing SGML documents. We also describe the design and implementation details of SGML document management system, which has capabilities of storing SGML instances, generating schema dynamically, and retrieving structure elements efficiently.

* 본 논문은 1997년도 시스템공학연구소(현재 한국전자통신연구원, 컴퓨터, 소프트웨어 기술 연구소)의 위탁과제로 수행된 결과임.
** 충남대학교 공과대학 컴퓨터공학과
*** 한국전자통신연구원 컴퓨터 · 소프트웨어 기술 연구소 자연언어처리 연구부
접수일자 1998년 9월 1일

1. 서론

다가오는 21세기는 고도화된 정보화 사회가 될 것이며, 이러한 정보화 사회에서 가장 필수적인 요소로는 기하급수적으로 발생되는 전자 문서를 관리하는 것이라 할 수 있다. 특히, 디지털 도서관(Digital Library), 인트라넷 상에서의 CSCW(Computer-Supported Cooperative Work), CALS(Commerce At the Light Speed) 등 대량의 전자 문서를 관리하거나 구축 해야 하는 요구가 발생하면서 대량의 문서 정보를 효율적으로 저장하고, 관리하는 시스템이 요구되고 있다.(Sachs-Davis, 1994)

SGML(Standard Generalized Markup Language)은 ISO에서 문서 표현 및 교환 표준으로 제정되었으며, 구조 정보로 사용할 수 있는 마크업(Markup)을 포함하고 있어 문서를 구조 단위로 표현할 수 있다. 이는 대량의 전자 문서를 저장하거나 관리하는데 적합하다.

SGML 문서에 관련된 다양한 응용 프로그램에 대한 지원과 문서의 공유 등을 지원하기 위해서는 데이터베이스에 기반한 SGML 문서 관리 시스템이 필수적으로 요구되며, SGML 문서 관리 시스템을 개발하기 위해서는 다음과 같은 네 가지 사항을 고려해야 한다.(Francois, 1996)

첫째는 SGML 문서의 데이터 모델로, 이는 SGML이 포함하고 있는 많은 시맨틱(semantic)들을 손실 없이 데이터베이스에 표현 할 수 있도록 모델링하는 것이다.

둘째는 새로운 문서 구조를 정의하고 있는 SGML DTD를 분석하여, 이에 대한 스키마를 데이터베이스에 생성하는 것으로, 이는 다양한 구조의 문서들을 효과적으로 관리 할 수 있는 기반이 된다.

셋째는 SGML 인스턴스를 데이터베이스에 효율적으로 저장하는 것으로, 새로운 SGML 인스턴스가 들어 오면, 이미 생성되어 있는 스키마에 이를 저장할 수 있어야 한다.

넷째는 데이터베이스에 저장된 SGML 인스턴스를 문서 또는 구조 단위로 꺼내는 것이다. 이는 한 문서의 공동저작이나 구조 단위의 검색을 요구할 때 필연적으로 요구된다.

본 논문에서는 위의 네가지의 기능을 모두 지원하는 SGML 문서 관리 시스템을 설계·구현 하였다.

본 논문의 구성은 다음과 같다. 2장에서는 SGML 문서의 효율적인 관리를 위해 설계한 SGML 문서의 데이터 모델에 대해서 기술하고 있으며, 3장에서는 SGML 문서 관리 시스템의 전체 구조와 이를 구성하는 각 모듈들의 기능에 대해서 기술하고, SGML 문서 관리 시스템의 기능 설계에 대해서 설명한다. 4장에서는 구현 환경과 SGML 문서 관리 시스템의 각 구성 모듈에 대한 구현 내용을 기술한다. 그리고 기존에 연구되어 온 SGML 문서의 데이터 모델링 방법들과 본 논문에서 제시한 방법을 5장에서 비교하여 설명한다. 마지막으로, 6장에서는 결론 및 향후 연구 내용에 대해 기술한다.

2. SGML 문서의 데이터 모델

SGML 문서 관리 시스템을 구현하기에 앞서 우선적으로 고려하고 설계되어야 하는 것이 SGML 문서의 데이터 모델이다. 본 장에서는 SGML 문서가 지닌 논리적 구조인 엘리먼트와 애트리뷰트, 엔티티의 효율적인 모델링 방안을 제시한다.

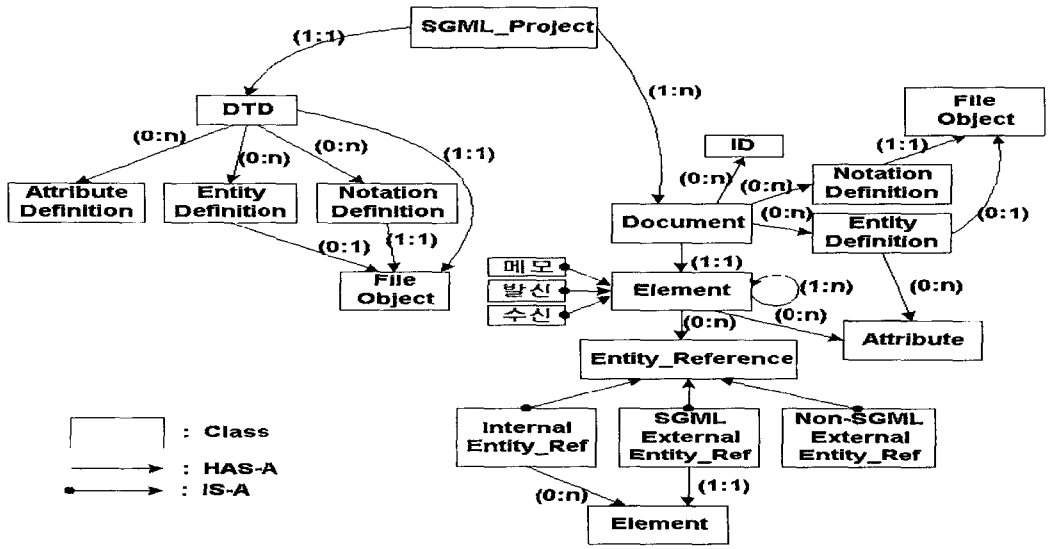
2. 1 SGML 문서의 논리적 구조 모델

SGML 문서의 논리적 구조 모델은 SGML 문서가 포함하고 있는 많은 특성들을 손실 없이 데이터베이스에 표현하여, 이들의 시맨틱을 기반으로 효율적으로 문서를 관리하고 검색하기 위함이다.

본 논문에서는 SGML 문서의 논리적 구조 모델을 DTD(Document Type Definition) 독립 스키마 모델(DTD-Independent Schema Model)과 DTD 의존 스키마 모델(DTD-Dependent Schema Model)로 나누어 설계하였다.

〈그림 1〉은 본 논문에서 설계한 DTD 독립 스키마 모델과 DTD 의존 스키마 모델을 모두 보여주는데, 음영으로 표현된 메모, 발신 그리고 수신은 DTD 의존 스키마 모델 부분이고 나머지는 DTD 독립 스키마 모델 부분이다.

DTD 독립 스키마 모델링의 구성은 엘리먼트 모델(element model), 애트리뷰트 모델(attribute model), 엔티티 참조 모델(entity reference model), SGML 프로젝트 모델(SGML project model), DTD 모델(DTD model), 문서 모델(document model)로 구성되어 있으며, 이러한 모델은 SGML DTD와 인스턴스 사이의 관계에 대한 정보나 DTD



〈그림 1〉 SGML 문서의 논리적 구조 모델링

자체에 정의된 애트리뷰트, 엔티티, 표시법에 대한 정보, 그리고 SGML 인스턴스 자체에 정의된 엔티티, 표시법 등의 포괄적인 정보의 관리를 목적으로 설계된 것이다. 또한 이것은 DTD 의존 스키마 모델링에서 새로운 DTD에 대한 클래스를 생성할 때 필요한 엘리먼트, 애트리뷰트, 엔티티 등의 기본 클래스 형을 포함하고 있다.

DTD 의존 스키마 모델은 특정 DTD에 의존해서 생성되는 클래스들의 집합으로, 데이터베이스내에서는 엘리먼트, 애트리뷰트, 엔티티 참조 클래스들을 이용해서 문서의 구조를 표현한다. 이때 사용되는 엘리먼트, 애트리뷰트, 엔티티 참조에 대한 클래스는 SGML DTD의 엘리먼트, 애트리뷰트, 엔티티 참조에 대한 특성을 기반으로 정의된 클래스로 DTD 독립 스키마 모델에 포함되어 있다.

DTD 의존 스키마 모델링에 대한 이해를 돕기 위해 간단한 문서 구조를 갖고 있는 메모 DTD〈그림 2〉를 예로 들어 설명하기로 한다.

〈그림 2〉는 메모 DTD 정의이고 〈그림 3〉은 메모 DTD의 문서 구조를 반영해서 본 시스템에서 생성된 클래스들로 구성된 DTD 의존 스키마이다. 〈그림 3〉의 클래스명은 엘리먼트 이름으로 생성되며, 각 클래스의 필드는 엘리먼트의 내용 모델(content model)에 대한 정보를 반영한다. 예를 들면, 메모 엘리먼트에 대응되는 클래스는 메모이며, 이

```
<!-- DTD for simple office memoranda                                    -->
<!--      ELEMENTS MIN CONTENT (EXCEPTIONS)                             -->

<!ENTITY % ISOdia  PUBLIC "ISO 8879:1986//ENTITIES Diacritical Marks//EN"  >
%ISOdia;
<!ENTITY % ISOnum  PUBLIC "ISO 8879:1986//ENTITIES Numeric and Special Graphic//EN" >
%ISOnum;
<!entity % test SYSTEM "c:\Program Files\SERI\Kleox\Entities\test.ent"     >
%test;
<!NOTATION bmp SYSTEM "BMP"                                               >
<!NOTATION cpp SYSTEM "CPP"                                               >
<!NOTATION drw SYSTEM "DRW"                                               >
<!ENTITY   CNU         "Chungnam National University"                       >
<!ELEMENT 메모  - - (( 수신 & 발신), 본문, 인사말?)
<!ELEMENT 수신  - O (#PCDATA)                                             >
<!ELEMENT 발신  - O (#PCDATA)                                             >
<!ELEMENT 본문  - O (P*)                                                  >
<!ELEMENT P  - O (#PCDATA|Q|그림)*                                        >
<!ELEMENT Q  - O CDATA                                                    >
<!ELEMENT 그림  - O EMPTY                                                 >
<!ELEMENT 인사말  - O RCDATA                                              >
<!ATTLIST  메모
          기밀        (confiden|public)     public
          버전          CDATA       #IMPLIED
>
<!ATTLIST  발신
          기관명        NAMES       #IMPLIED
>
<!ATTLIST  그림
          filename  ENTITY  #IMPLIED                                      >
```

〈그림 2〉 메모 DTD

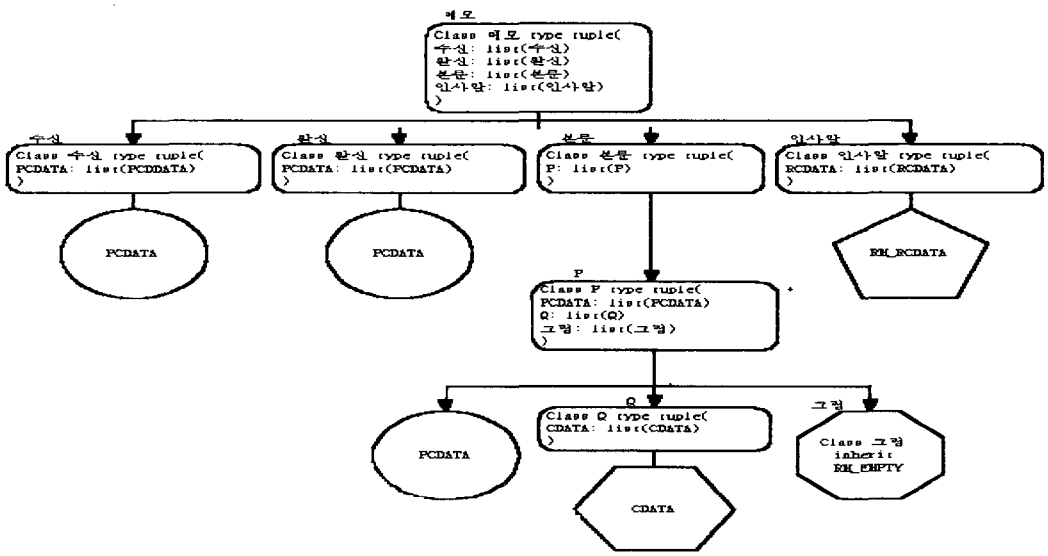
클래스를 구성하고 있는 필드들은 하위 엘리먼트에 대응되는 클래스의 리스트 형으로 선언되어 있다.

〈그림 3〉에서 직사각형으로 표현된 클래스는 〈그림 4〉에서 정의된 엘리먼트 클래스 중 NT를, 원으로 표현된 클래스는 PCDATA를, 오각형으로 표현된 클래스는 RCDATA를, 육각형으로 표현된 클래스는 CDATA를, 8각형으로 표현된 클래스는 EMPTY를 각각 계승하여 생성된 클래스이다.

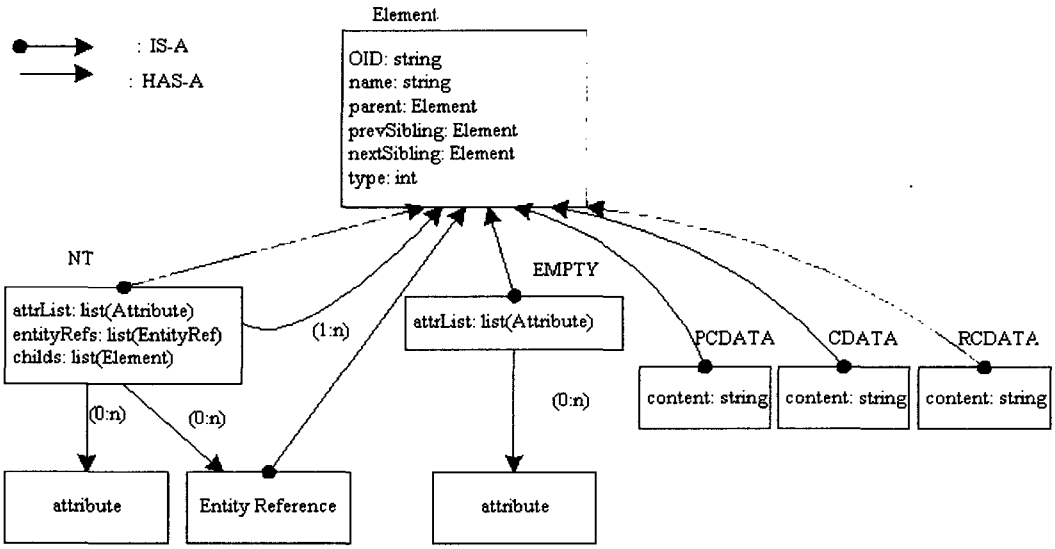
2. 2 엘리먼트 모델링(Element Modeling)

SGML에서 엘리먼트는 문서의 구조를 정의할 때 구조 표현의 단위로 사용되는 아주 중요한 구성 요소이다. 본 논문에서는 엘리먼트를 크게 단말 엘리먼트(terminal element)와 비단말 엘리먼트(non-terminal element)로 나누어 모델링을 하였다. 전자는 SGML DTD에서 서브 엘리먼트를 포함하지 않는 엘리먼트 즉, PCDATA, RCDATA 혹은 CDATA와 같은 SGML 데이터 타입이나 EMPTY 타입으로 선언된 엘리먼트를 의미하고, 후자는 서브 엘리먼트나 서브 엘리먼트 리스트를 포함하는 엘리먼트를 의미한다

〈그림 4〉에서 엘리먼트 클래스(Element Class)는 단말 엘리먼트와 비단말 엘리먼트가 공통적으로 갖는 특성만을 가지고 만든 클래스로 이 클래스는 OID는 객체 식별자이며 문서 트리에서의 위치 정보를 위한 parent, prevSibling, nextSibling 필드와 항해에 관한 연산을 갖는다. 비단말 엘리먼트 클래스(NT Element Class)는 SGML 애트리뷰트, 엔티티 참조, 자식 노드의 순서 정보와 SGML 애트리뷰트 삽입을 위한 연산을 포



〈그림 3〉 메모 DTD에 대한 DTD 의존 스키마



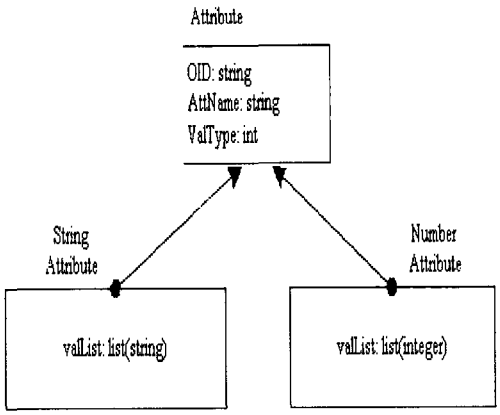
〈그림 4〉 엘리멘트 모델

함한다. 단말 엘리먼트 클래스들인 PCDATA, CDATA, RCDATA는 하위 엘리먼트를 갖지 않고, 실제 각 엘리먼트의 내용을 갖는다. 따라서 이 클래스는 내용 저장을 위한 content 필드와 데이터의 삽입, 삭제 연산을 갖는다. EMPTY는 하위 엘리먼트와 엔티티 참조도 갖지 않고, 내용도 포함하지 않는다. 단지 attribute 리스트만을 갖는데 이를 위해 EMPTY 클래스는 attList 필드와 SGML 애트리뷰트 삽입연산을 갖는다.

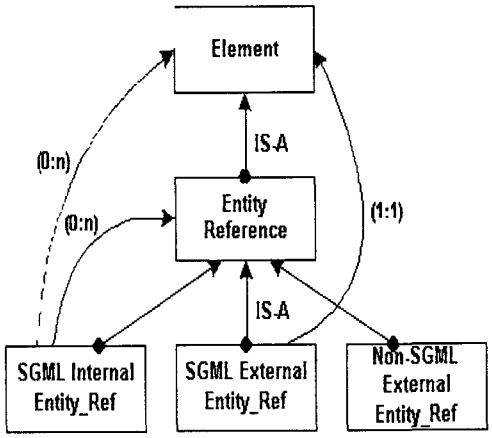
2. 3 애트리뷰트 모델링 (Attribute Modeling)

애트리뷰트는 특정 엘리먼트의 속성을 정의하는 것이므로 애트리뷰트 이름과, 타입, 값에 대한 관리가 필요하다. 본 논문에서는 애트리뷰트 모델링을 선언 값(Declared value)에 따라 크게 두 가지로 나누었다. 하나는 선언 값이 NUMBER나 NUMBERS로 선언된 경우로 애트리뷰트 값이 숫자로 관리되어야 하는 경우이며, 다른 하나는 선언 값이 NUMBER와 NUMBERS를 제외하고 선언된 경우로 애트리뷰트 값이 문자열 형태로 관리 되어야 하는 경우이다.

애트리뷰트 클래스(Attribute Class)는 애트리뷰트의 이름과 선언 값을 위한 AttName, ValType 필드와 SGML 애트리뷰트의 이름, 선언 값의 삽입, 삭제, 변경 연산을 포함한다. 숫자 애트리뷰트 클래스(Number Attribute Class)는 애트리뷰트의 값을 숫자로 관리하기 위해 integer형의 리스트 형으로 선언된 ValList 필드와 애트리뷰트 값의 삽입, 삭제, 변경 연산을 갖는다. 또한 문자열 애트리뷰트 클래스(String Attribute Class)는 애트리뷰트의 값을 문자열



〈그림 5〉 애트리뷰트 모델



〈그림 6〉 엔티티 모델

로 관리하기 위해 string형의 리스트로 선언된 ValList 필드와 애트리뷰트 값의 삽입, 삭제, 변경 연산을 갖는다.

2. 4 엔티티 모델링(Entity Modeling)

엔티티는 〈그림 6〉과 같이 세가지 경우를 고려할 수 있다. 엔티티가 같은 DTD 내부에 선언되고 동시에 참조까지 되는 경우와 DTD 외부에 저장되어 참조되는 경우이며, DTD 외부에서 참조되는 경우는 다시 그 엔티티 텍스트가 SGML 데이터인지, Non-SGML 데이터인지에 따라 구분할 수 있다. 엔티티는 엘리먼트의 서브 클래스로 생성된다.

3. SGML 문서 관리 시스템의 설계

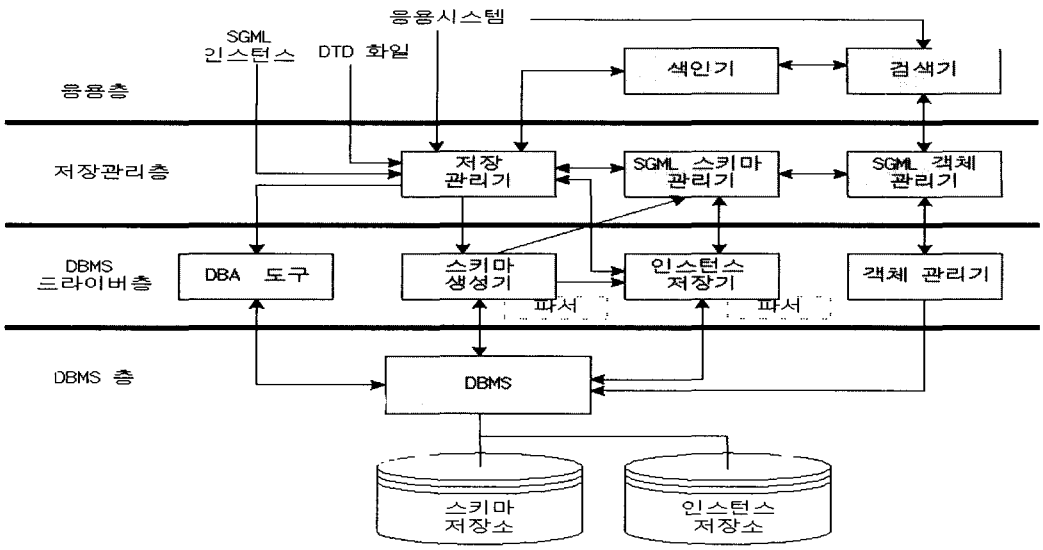
본 장에서는 SGML 문서 관리 시스템의 설계에 대해서 기술한다. 1절에서는 SGML 문서 관리 시스템에 필요한 모듈들을 설계하고, 각 모듈의 기능에 대해서 기술한다. 2절에서는 SGML 문서 관리 시스템의 기능 설계에 대해 기술한다.

3. 1 SGML 문서 관리 시스템의 구조

〈그림 7〉은 본 논문에서 설계한 SGML 문서 관리 시스템의 전체 구조를 보여준다.

SGML 문서 관리 시스템의 전체 구조는 응용 층(application layer), 저장관리 층, DBMS 드라이버 층(DBMS driver layer), DBMS 층(DBMS layer)의 4개의 층으로 구성된다. 특히, 전체구조에서 각 모듈을 층별로 나누어 설계한 이유는 분산 환경과 서로 다른 종류의 DBMS를 저장소로 사용할 수 있게 하기 위함이다.

DBMS 층은 SGML DTD에 대한 정보를 저장하는 스키마 저장소와 특정 인스턴스를 저장하는 인스턴스 저장소로 구성되며, 저장소로 사용되는 DBMS들은 분산, 이종으로



〈그림 7〉 SGML 문서 관리 시스템의 전체 구조

구성될 수 있다.

DBMS 드라이버 층은 저장소로 사용하는 DBMS의 종류에 대한 투명성을 제공한다. 이 층은 DTD 독립 스키마를 생성하는 DBA 도구(DBA Tool), 특정 DTD에 대한 스키마를 생성하는 스키마 생성기, 인스턴스가 입력되었을 때 DBMS에 이를 저장하는 인스턴스 저장기, DBMS에 저장된 SGML 객체를 관리하는 객체 관리기로 구성된다. 그리고, DBMS 드라이버 층은 특정 DBMS 마다 이에 의존적으로 구현되므로, 각 DBMS 마다 이에 맞는 드라이버 층을 구현해야 한다.

저장관리 층은 DBMS 드라이버 층을 관리하는 기능을 수행한다. 저장관리 층의 상위에 존재하는 모듈들에게 저장관리 층을 포함한 하위 층들이 포함하고 있는 각 모듈들에 대한 통합 인터페이스를 제공하는 저장관리기, SGML DTD, SGML 인스턴스, 분산된 저장소에 대한 포괄적인 정보를 관리하는 SGML 스키마 관리기, 각 저장소의 SGML 객체를 DBMS 드라이버 층의 객체 관리기를 통해 관리하는 SGML 객체 관리기로 구성된다.

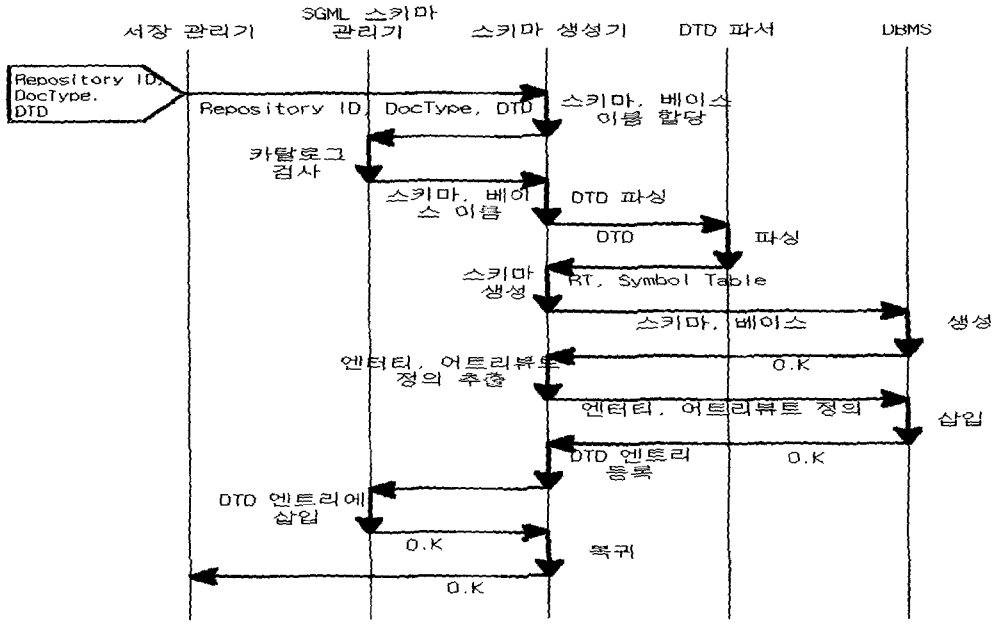
응용 층은 DTD 파일, 인스턴스 파일, 여러 응용 프로그램들과 사용자 등으로 구성된다.

3. 2 SGML 문서 관리 시스템의 기능 설계

본 절에서는 SGML 문서 관리 시스템의 주요한 기능을 기반으로 각 모듈간의 인터페이스 관계와 데이터 처리 흐름도를 기술한다.

3. 2. 1 DTD 의존 스키마 생성

새로운 SGML 인스턴스를 저장하기 위해

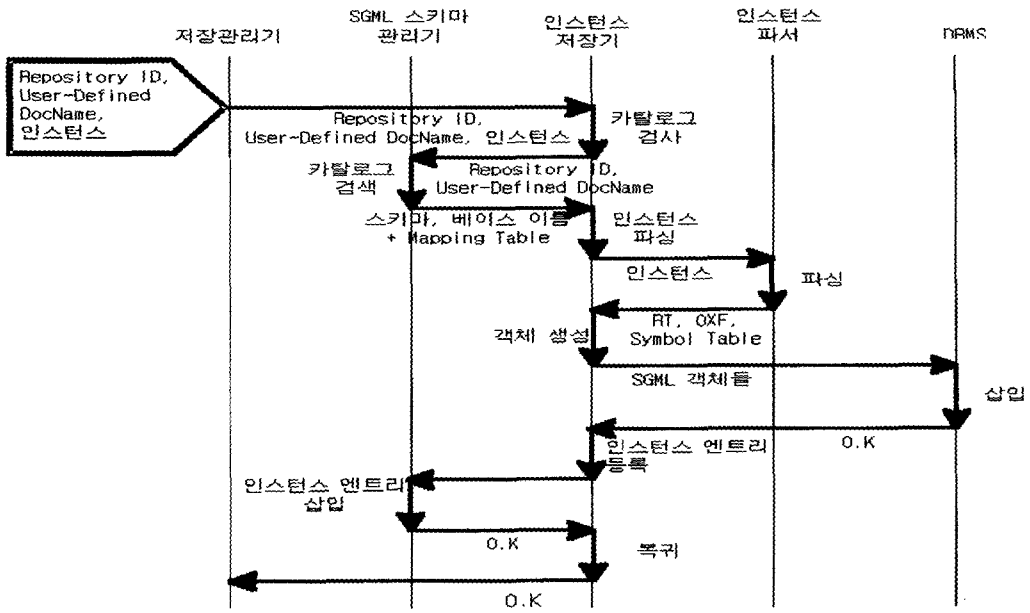


〈그림 8〉 새로운 스키마 생성 과정

서는 새로운 스키마의 생성이 필요하다. 〈그림 8〉은 이와 같은 새로운 스키마 생성 과정을 보여준다. 새로운 스키마를 만들기 위해 저장 관리기가 특정 DBMS 드라이버 층에 있는 스키마 생성기의 API를 호출하면, 스키마 생성기는 스키마 관리기를 호출해서 지금 생성할 스키마가 이미 존재하는 지를 검사하고 존재하지 않는 것이 확인되면 스키마와 베이스 이름을 할당 받는다. DTD 파서를 이용해서 해당 DTD를 파싱하고, 스키마 생성시 필요한 정보와 DTD 정보로 관리되어야 하는 정보를 추출한다. 이렇게 추출한 정보를 이용해서 스키마를 생성하고, 관리되어야 할 DTD 정보는 해당 클래스에 저장한다. 마지막으로 DTD에 대한 정보를 스키마 관리기에 등록한다.

3. 2. 2 객체 생성

저장 관리기에 SGML 인스턴스가 들어오면 특정 DBMS 드라이버 층의 인스턴스 저장기를 호출해서 제어권을 넘긴다. 〈그림 9〉는 이와 같은 과정에 대한 그림이다. 인스턴스 저장기는 인스턴스를 저장할 스키마가 생성되어 있는지를 검사하고, 생성되어 있으면 스키마와 베이스 이름, 그리고 엘리먼트 명에 클래스 명을 대응시킨 테이블을 전달 받는다. 인스턴스를 저장하기 전에 인스턴스 파서를 이용하여 문제가 없는지 검증하고, 없을 때 나오는 결과를 가지고 SGML 객체를 생성한다. 마지막으로 인스턴스에 대한 정보를 스키마 관리기에 등록하는데, 이 정보를 이용하여 색인기가 아직 색인되지 않은 문서를 구분하고 색인하게 된다.



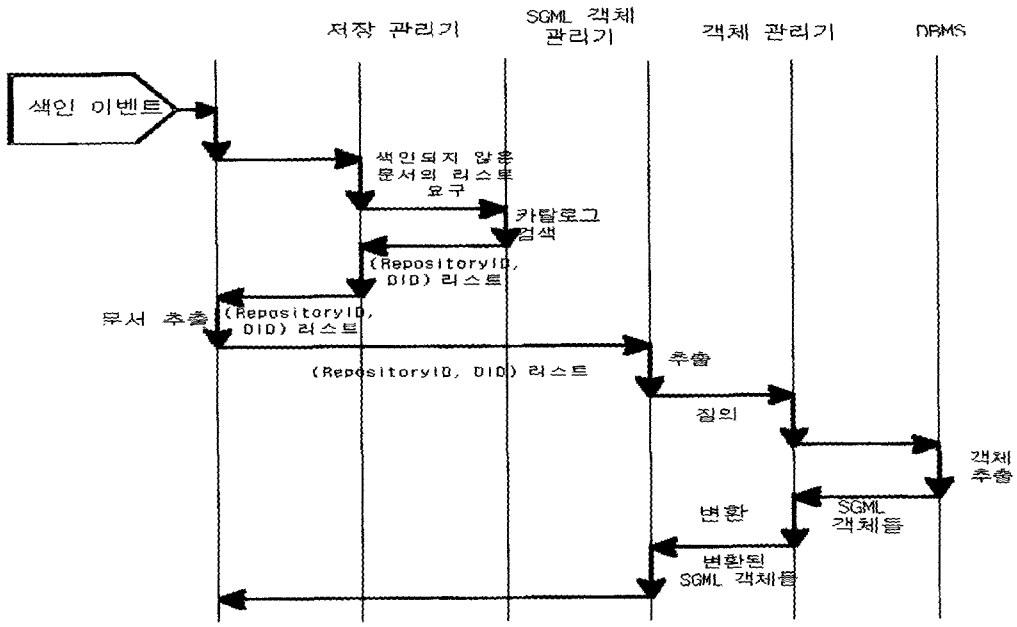
〈그림 9〉 객체 생성과정

3. 2. 3 색인을 위한 문서 Fetch

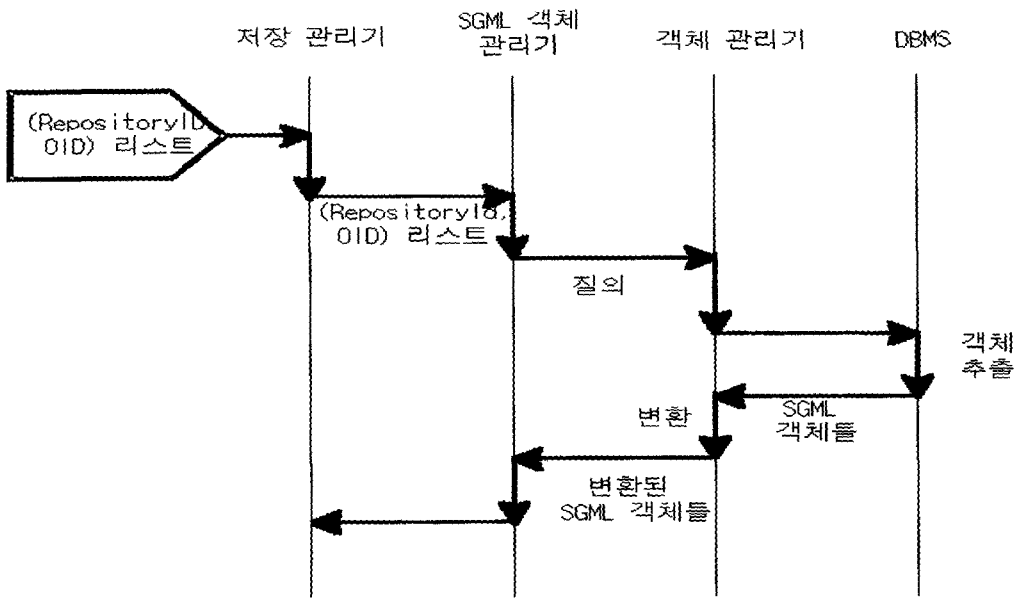
일반적으로 색인은 관리할 문서를 저장하기 전에 수행되어 구성되지만, 본 논문에서는 일단 SGML 인스턴스를 먼저 저장하고, 색인 수행이 필요할 때 색인을 한다. 따라서 색인 이벤트가 발생하면 색인기는 저장 관리기에게 저장된 인스턴스들 중 색인 해야 할 문서들의 리스트를 요청한다. 저장 관리기는 SGML 스키마 관리기에게 이를 요청해서 Repository ID와 DID의 리스트를 색인기에게 넘겨주고, 색인기는 이를 다시 SGML 객체 관리기에게 넘겨준다. SGML 객체 관리기는 해당 DBMS 드라이버 층의 객체 관리기에 적절한 질의를 전달하고, 객체 관리기는 이의 수행 결과로 인스턴스를 꺼내준다. 〈그림 10〉은 이와 같은 과정을 보여준다

3. 2. 4 SGML 객체 Fetch

사용자가 원하는 정보를 얻기 위해 질의를 하면 검색기에서는 질의문을 분석하여 색인기에서 사용자에게 결과로 반환할 SGML 객체의 Repository ID와 OID의 리스트를 얻어온다. 그리고 검색기는 색인기로부터 받은 Repository ID와 OID의 리스트를 저장 관리기에게 전달하고, 저장 관리기는 다시 SGML 객체 관리기로 전달한다. SGML 객체 관리기는 각 DBMS 드라이버에 있는 객체 관리기에게 적절한 질의를 전달하고, 객체 관리기는 수행 결과를 반환한다. 이와 같은 과정은 〈그림 11〉과 같다.



〈그림 10〉 색인을 위한 문서 Fetch



〈그림 11〉 SGML 객체 Fetch

4. SGML 문서 관리 시스템의 구현

본 장에서는 SGML 문서 관리 시스템의 구현 환경과 각 모듈의 구현 내용에 대해서 기술한다.

4. 1 구현 환경

〈그림 12〉는 SGML 문서 관리 시스템의 구현 환경을 나타낸다. SGML 문서 관리 시스템의 운영 체제는 SunOS 2.5.1을 기반으로 하였고, 저장소로 사용할 데이터베이스는 O2를 사용하였다. O2는 객체지향 데이터베이스 시스템(Object-Oriented DBMS)으로 계승(inheritance) 개념이나 메소드(method) 등과 같은 객체 지향 개념의 여러 특성을 지원하며, O2에서 제공하는 질의어 OQL은 객체 지향 데이터베이스의 표준인 ODMG '93을 따르고 있다[10, 11].

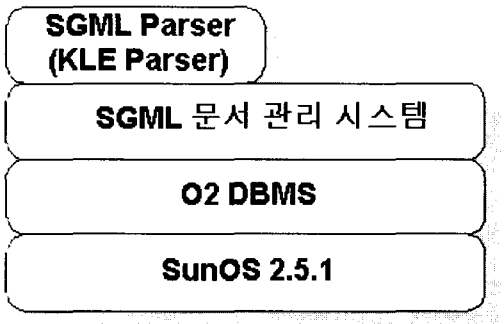
SGML 파서는 시스템 공학 연구소에서 개발한 KLE 파서를 사용하였다. 이는 SGML DTD와 인스턴스의 적합성을 검증하고, 적합하다고 검증된 SGML DTD와 인스턴스에 대해서는 파싱 결과로 RT(Rule Tree)와 OXF 트리 그리고 심볼 테이블들을 반환한다. RT는 DTD에 정의된 엘리먼트를 기반으로 트리 형태로 표현한 것이고, OXF 트리는 인스턴스에 있는 마크업(Markup)을 기반으로 트리 형태로 표현한 것인데, 이들은 SGML 문서를 처리하는 응용 프로그램 개발에서 중간 데이터 형태로 사용된다.

〈그림 13〉은 메모 DTD에 대한 인스턴스의 예를 보여주고, 〈그림 14〉는 〈그림 13〉을 KLE 파서를 이용해서 파싱 했을 때 생성되는 OXF를 보여준다.

〈그림 14〉에서 직사각형은 SGML 데이터 타입인 #PCDATA, RCDATA, CDATA인 경우를 표현한 것이고, 애트리뷰트에 대한 표현과 RT와 관련된 링크의 표현은 생략하였다.

4. 2 객체 식별자(OID) 부여 및 관리

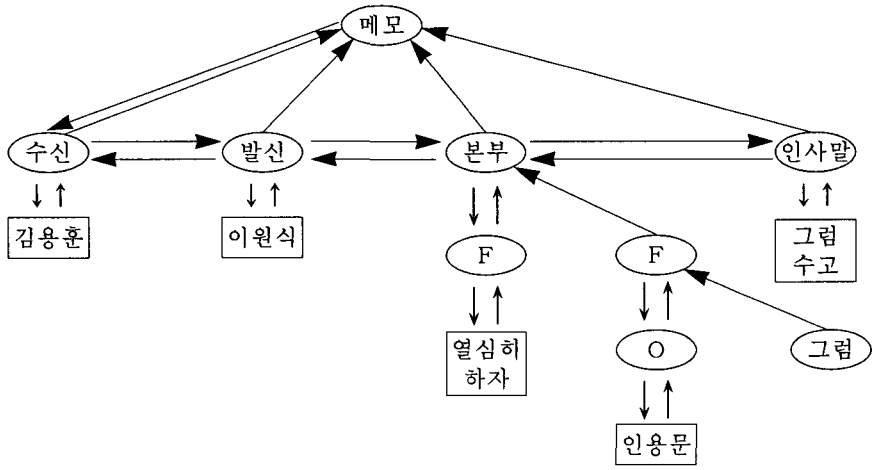
ODMG 객체 모델에서 모든 객체는 유일한 식별자를 가지며, 일반적으로 다른 객체를 참조하기 위해 사용된다. 그러나 ODMG



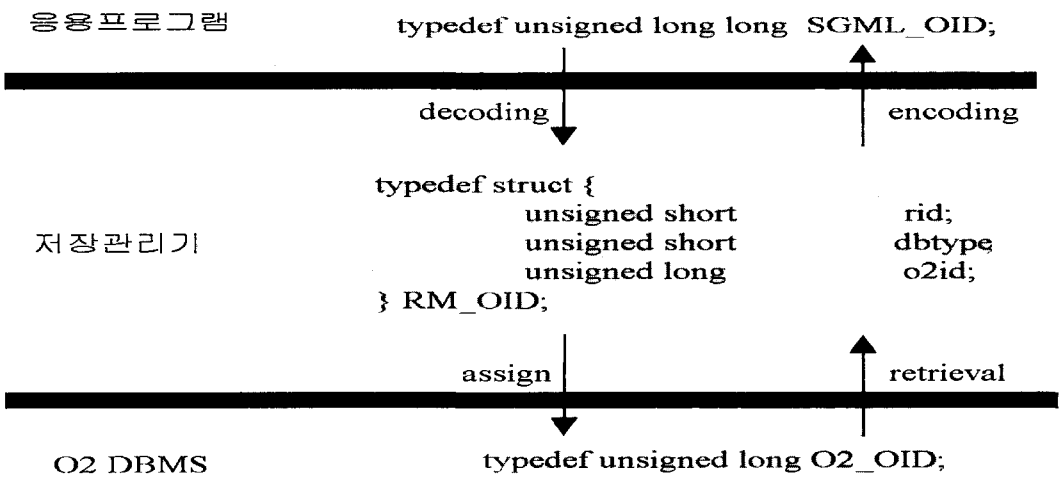
〈그림 12〉 시스템 구현 환경

```
〈!DOCTYPE 메모 SYSTEM [
〈!ENTITY 모델 SYSTEM "model.cpp" CDATA BMP 〉 ]〉
〈메모 기밀="PUBLIC"〉
  〈수신〉 김용훈 〈/수신〉
  〈발신 기관명="충남대학교 컴퓨터공학과"〉 이원석 〈/발신〉
  〈본문〉 〈P〉 열심히 하자. 〈/P〉
        〈P〉 〈Q〉 인용문 〈/Q〉 〈그림 FILENAME="모델"〉 〈/P〉 〈/본문〉
  〈인사말〉 그럼 수고. 〈/인사말〉
〈/메모〉
```

〈그림 13〉 메모 DTD에 대한 인스턴스의 예



〈그림 14〉 파싱 결과로 생성되는 OXF 트리의 예



〈그림 15〉 각 계층에서의 객체식별자

에서는 응용프로그램에서 데이터베이스에 저장된 객체를 접근하는데 필요한 객체 식별자에 대해 정의하고 있지 않다. 하부 저장 시스템인 O2 DBMS 4.6버전의 경우에도 물리적인 객체식별자를 제공하지 않으므로 O2 DBMS를 이용하는 저장 관리기는 저장 계층, 저장 관리기 계층, 저장 관리기 응용 계층에서 사용할 수 있는 객체 식별자를 할당하고 관리 할 수 있어야 한다. 이를 위해 저장 관리기는 각 계층에서 사용하는 식별자를 부여하여 사용한다. 다음의 〈그림 15〉는 각 계층에서 사용하는 객체식별자의 형태를 나

타낸다.

SGML_OID는 구조 정보 검색 서버에서 저장된 엘리먼트를 액세스하기 위해 색인 파일에 저장된다. RM_OID는 다수의 저장소 및 ODBMS를 구분하기 위한 필드 및 실제 ODBMS에 저장되는 객체의 식별자로 구성되며, O2_OID는 SGML 객체의 애트리뷰트로 O2 DBMS에 저장된다.

4. 3 저장 관리기 구현

저장관리기는 네트웍상의 외부 응용 프로그램의 요청에 대한 세션(session) 개방 및 종료와 저장시스템 내부 모듈에 대한 일의 제어 및 메시지 분배를 수행한다. 저장 관리기는 연결 지향형(connection-oriented) 서버이므로 클라이언트의 요청에 따른 상태 정보를 관리한다.

4. 4 SGML 스키마 관리기 구현

SGML 스키마 관리기의 역할은 다음과 같다. 첫째, 저장관리기의 요청에 따라 저장시스템 내부 모듈들의 요구에 대한 저장소 이름과 문서 이름에 대한 데이터베이스 엔티티로의 매핑(mapping) 서비스를 제공한다. 둘째, 색인되지 않은 문서 목록들을 유지하며 구조 정보 검색 색인기가 색인되지 않은 문서의 목록을 요구하는 경우에 해당 정보를 반환하며, 색인 여부를 갱신한다. 셋째, 새롭게 SGML 저장소가 생성될 때 저장소 식별자를 부여한다. SGML 스키마 관리기는 저장관리기와 메시지 큐를 통해 메시지를 전송하며, 데이터는 공유메모리를 통해 전달한다.

4. 5 SGML 객체 관리기 구현

SGML 객체 관리기는 데이터베이스에 엘리먼트 단위로 나뉘어서 저장된 객체를 객체 관리기로부터 건네받아 클라이언트가 요구하는 형태로 문서를 재구성하여 제공한다. 또한 구조 검색 서버로부터 fetch를 요구하는 객체와 문서의 식별자를 해석하여 ODBMS의 유형, 스키마, 데이터베이스를 결정하여 객체를 반환한다. 반환되는 객체의 형태는 객체의 종류나 클라이언트의 요구에 따라 달라지며 주로 엘리먼트 서브 트리, 특정 엘리먼트의 간략 정보, 문서 전체 트리나 익스포트(export) 형태, DTD정보 등을 반환한다. 엘리먼트나 문서 객체 및 DTD구조 정보를 반환할 때는 해당 객체의 트리 구조를 유지한 채 클라이언트에서 직접 접근 가능할 수 있도록 제공한다.

4. 6 스키마 생성기 구현

DTD 의존 스키마는 이미 데이터베이스에 등록된 DTD 독립 스키마를 토대로 만들어진다. DTD로부터 새로 생성되는 클래스는 DTD의 각 GI클래스로서 DTD파서를 통해 파싱한 후 얻어지는 내부 심볼 테이블을 이용하여 특정 DTD에 의존적인 스키마를 생성한다. 스키마 생성기는 클래스의 생성 뿐 아니라 DTD에 정의된 엔티티, 애트리뷰트, Notation에 대한 정의 항목들을 저장하는 역할을 수행한다. 즉 SGML_Repository 및

DTD에 대한 객체를 만들고 이를 저장하기 위해 베이스를 생성한다.

입력된 DTD화일마다 하나의 SGML저장소 즉, DTD마다 독립된 하나의 스키마와 이에 기초한 하나의 베이스가 생성된다. 스키마 생성기는 저장 관리기로부터 DTD화일 이외에 생성할 스키마 이름, 베이스 이름 등을 입력 인자로 받아 해당 DTD에 기초한 모든 문서를 저장할 SGML저장소의 기본 틀(스키마) 및 저장소 객체 자체를 생성하여 문서 인스턴스를 저장하기 위한 기반을 마련한다. SGML 저장소의 생성은 크게 3단계로 이루어진다.

첫째, 전처리 단계로서 입력된 DTD화일을 파싱한 후, 그 결과물인 RT(Rule Tree)를 이용하여 각 GI마다 해당 content model에 근거한 자식 엘리먼트의 구성요소를 추출하고, GI를 클래스 이름으로 바로 쓸 수 없으므로(특히, GI가 한글인 경우에 클래스 이름으로 한글을 쓸 수 없다.) GI별 클래스 이름을 부여한 테이블을 생성한다.

둘째, 스키마 생성단계로서 이미 데이터베이스에 있는 DTD 독립 스키마의 클래스로부터 상속 관계를 갖는 DTD 의존 스키마를 생성한다. DTD에 존재하는 모든 GI에 대해 독립된 클래스를 형성하며, 이들은 NT(Non-Terminal Element), ANY, EMPTY 엘리먼트 중의 하나이므로 이 세가지 클래스 가운데서 상속받는다. 어떤 클래스와 상속 링크를 갖는 지는 이미 전처리 단계에서 RT로 부터 얻어낸 심볼 테이블에 근거하여 결정된다.

마지막으로 SGML저장소의 기본 틀을 구축하는 단계로서 스키마 생성 단계에서 마련된 특정 DTD 스키마에 근거하여 베이스를 생성하는 한편, 실제로 SGMLRepository 객체 및 DTD에 정의된 엔티티, 애트리뷰트, Notation에 대한 정의 항목들을 저장한다.

4. 7 인스턴스 저장기 구현

인스턴스 저장기는 저장 관리기로부터 인스턴스를 받아 저장하는 모듈로서 실제로는 InsertInstance()라는 API를 제공하는 라이브러리 형태로 존재한다.

인스턴스 저장기에서 제공하는 API인 InsertInstance의 구현 알고리즘에 대한 설명은 다음과 같다.

1) 인스턴스 저장기는 o2_init() 라는 API를 이용해 O2 DBMS에 연결을 하게 된다. 저장 관리기로부터 입력된 스키마 이름과 베이스 이름을 이용하여 O2의 저장소를 열고, 스키마 관리기를 호출하여 Class 매핑 테이블을 받아온다.

2) 인스턴스 저장기는 인스턴스 파서를 이용하여 새로 들어온 인스턴스를 파싱하고, 그 결과로 RT와 OXF 그리고 심볼 테이블들을 받는다.

3) 새로운 인스턴스의 저장을 위하여 Document 클래스 형의 객체를 만들고, 이 객체의 Did와 docID 필드에 각각 새로운 인스턴스의 DID와 파일명을 넣는다.

4) 파싱 결과로 받은 테이블에서 Entity Definition과 Notation 정보를 추출하여, EntityDefinition과 NotationDefinition 클래스

형 객체들에 저장하고 그 핸들값들을 3)에서 생성한 객체의 entityDecl과 notatDecl 필드에 넣는다. EntityDefinition이나 Notation의 정보들 중 외부 파일명이 있는 경우 FileObj 클래스 형 객체를 생성하여 파일의 내용을 저장하게 된다. 이 객체의 핸들은 EntityDefinition이나 NotationDefinition 클래스 형 객체들의 FileObj라는 필드에 넣는다.

5) 인스턴스 저장기는 파싱의 결과로 받은 OXF 트리의 루트부터 시작해서 전 노드들을 순회하며 객체들을 생성하고, 각각의 필드들의 값들을 채우게 된다.

6) 트리의 순회로 모든 객체 생성이 끝나면 루트 엘리멘트를 3)에서 생성했던 Document 클래스 형 객체의 rootElem 필드에 넣는다. 그리고 이 Document 클래스 형 객체는 Persistent Root의 docList 필드에 추가된다.

7) o2__validate()라는 O2 Engine API를 통해서 생성된 객체들이 O2 DBMS에 저장된다.

8) 마지막으로 인스턴스는 새롭게 생성된 인스턴스의 DID를 저장관리기로 넘겨주게 된다.

4. 8 객체 관리기 구현

객체 관리기는 인스턴스 저장기에 의해서 O2 DBMS에 저장된 SGML 객체를 꺼내는 기능을 맡고 있는 저장 관리기의 한 모듈로, 색인기나 검색기에서 O2 DBMS에 저장된 SGML 객체의 접근이나 정보를 요구때 사용할 수 있는 API들을 제공한다. 실제로는 라이브러리 형태로 존재한다. 객체 관리기에서 제공하는 API의 구현 알고리즘에 대한 설명은 다음과 같다.

〈그림 16〉은 객체 관리기에서 제공하는 모든 API를 나타내는데, 이들 중 Open과 Close는 세션과 스키마, 베이스를 오픈하고 종료하는 함수이다.

◆ GetAllElementList

특정 DTD에 있는 모든 엘리먼트 명을 넘겨주는 API로, 이는 검색기에서 사용자에게 엘리먼트에 대한 정보를 넘겨줄 때 사용된다.

◆ GetDocument

문서의 식별자인 DID(Document ID)를 받아서 O2 DBMS에서 이에 해당되는 문서를 꺼내오는 API로 색인기나 검색기에서 사

```
Void Open(const char *schema, const char *base);
Void Close();

Int GetAllElementList (const char *filename);
Int GetDocument(const REPOSITORY__ID &rid, const O2__OID &did, const char *filename);
Int GetElement(const REPOSITORY__ID &rid, const O2__OID &oid, const char *filename);
Int GetBriefInfo(int num, const O2__OID *oidlist, const char *filename);
Int GetFileObjectInDtd(const RM__FILETYPE &, char*);
```

그림 16. 객체 관리기의 API

용한다. 본 함수는 O2 DBMS에 저장된 SGML 객체를 검색기나 색인기에게 주는 포맷으로 단순 OXF 트리 형태를 사용한다. 단순 OXF 트리는 KLE 파서가 인스턴스를 파싱했을 때 메모리에 생성되는 OXF 트리를 간소화한 형태이다. 이 같은 형태를 만들어 주는 이유는 SGML 객체가 가지고 있는 여러가지 특성들을 그대로 보존하면서 전달할 수 있는 좋은 방법이기 때문이다. 만일 SGML 문서의 형태로 반환을 한다면 이를 파싱해서 사용해야 하는 단점이 있다. 본 함수의 구현 알고리즘은 다음과 같다.

1) 입력으로 들어온 DID를 받아서 대상 SGML 객체 탐색의 시작점이 되는 루트 엘리먼트의 핸들을 받아오는 OQL을 생성하고 실행한다. 예를 들면, Persistent root 명은 RM__root__Memo__b, DID는 3이라고 가정했을 때 문서의 루트 엘리먼트의 핸들을 받아오는 OQL은 다음과 같이 표현된다.

```
Select d.rootElem from d in RM__root__
Memo__b.docList where d.o2id = 3
```

2) 받아온 루트 엘리먼트의 핸들을 이용해서 SGML 객체를 탐색하며, 인스턴스 저장기가 부여한 OID와 단순 OXF 트리에서 사용할 ID를 매핑시키는 테이블을 생성하고, 총 노드의 수를 계산한다.
3) 루트 엘리먼트에서 부터 다시 탐색을 하면서 2)에서 만든 매핑 테이블과 노드의 수를 이용해서, 단순 OXF 트리를 구성할 때 사용되는 정보를 파일 형태로 쓴다.
4) 파일을 읽어들여 메모리에 단순 OXF 트리를 구성한다.

◆ GetElement

인스턴스 저장기가 SGML 문서를 O2 DBMS에 저장할 때는 문서 단위로 저장이 이루어 지지만 꺼낼 때는 문서 전체와 문서의 일부분 모두 가능하다. GetElement는 문서의 일부분인 SGML 객체를 꺼내는 기능을 제공하는 API로 검색기에서 사용한다.

입력으로 원하는 SGML 객체의 OID를 받고 GetDocument와 같이 단순 OXF 트리를 구성하는 정보를 파일로 생성한다.

◆ GetBriefInfo

검색기는 질의를 처리해서 얻은 SGML 객체에 대한 정보를 사용자에게 보여주고, 이를 보고 사용자는 자신이 원하는 객체를 선택한다. GetBriefInfo는 결과로 나온 OID에 대한 메타데이타를 꺼내오는 기능을 한다.

그러나, 아직 어떤 부분을 요약정보로 사용할지, 그리고 어떻게 요약정보로 사용할 부분을 지정할지 등의 아직 결정하지 못한 부분들이 많이 있다. 따라서 주어진 OID의 서브 트리중 내용을 담고 있는 첫번째 노드의 것을 가져오는 것을 GetBriefInfo API의 기능으로 정의를 했고 이를 구현하였다. GetBriefInfo가 처리해야 할 경우와 결과 파일 형태는 다음과 같다.

1) 입력으로 들어온 OID에 대한 객체가 존재하지 않을 때, OID를 쓰고 길이는 1로 쓴다

2) 입력으로 들어온 OID에 대한 객체가 존재하고, 이 객체가 포함하고 있는 노드 중 적어도 하나가 내용을 갖고 있는 경우, OID를 쓰고, 내용의 길이를 쓰고, 내용을 쓴다.
3) 입력으로 들어온 OID에 대한 객체가 존재하고, 이 객체가 포함하고 있는 노드 중 내용을 갖고 있는 노드가 없을 때, OID를 쓰고, 길이를 0으로 쓴다.

◆ GetFileObjectInDtd

DTD의 구조 정보를 트리 형태로 갖고 있는 RT 파일이나 DTD 텍스트 파일 자체를 반환하는 함수로서 입력 인자에 따라 바이너리 파일 포멧의 RT 또는 아스키 파일 형태인 DTD파일을 생성해준다.

5. 관련연구

일반적으로 SGML 문서의 데이터 모델링은 SGML 문서의 효율적인 저장과 관리에 기본이 되는 것이다.(Sachs-Davis, 1994; Christophides, 1994) 본 논문에서는 SGML 문서의 시맨틱을 그대로 유지하면서 이를 효과적으로 관리할 수 있는 SGML 문서의 데이터 모델링에 대한 방법을 제안하였다. 이 장에서는 지금까지 연구되어 온 기존의 방법과 본 논문에서 제시한 방법을 비교하여 기술한다.

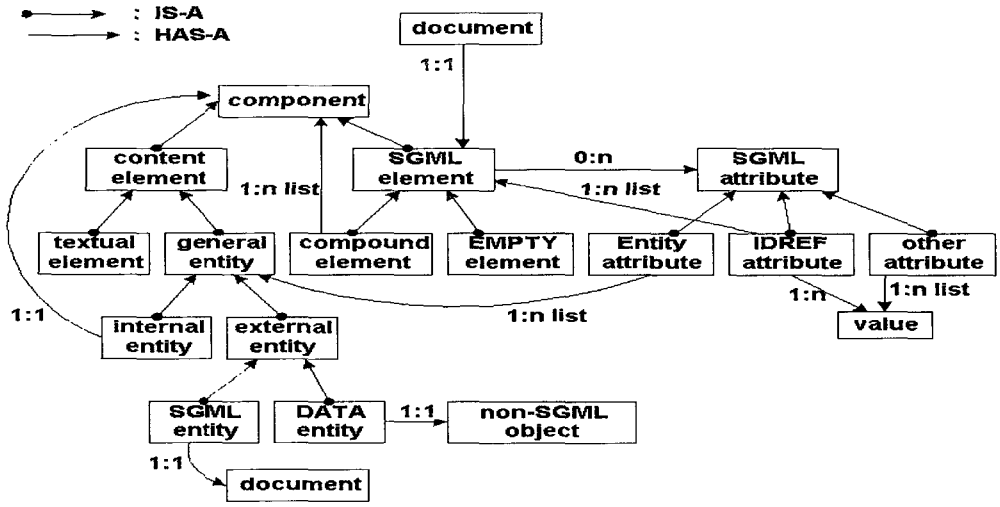
지금까지 제시되 온 모델링을 보면 관계모델(relational model), 확장된 관계 모델(extended relational model), 복합 객체 모델(complex object model), 객체 기반 모델(object-based model), 엘리먼트 기반 모델(element-based model) 등으로 나눌 수 있다.

관계모델은 관계형 데이터베이스를 기반으로 하므로 문서의 구조에 대한 충분한 정보를 유지하기 위해서 필요한 테이블과 튜플의 수가 기하급수적으로 늘어난다는 단점이 있는 반면 안정화 되어있는 관계형 데이터베이스 시스템이 제공해주는 모든 질의 능력을 활용할 수 있다는 장점이 있다. 관계모델에서 나타나는 문제점을 해결하기 위해 nesting, reference를 지원하는 확장된 관계 모델이나 복합 객체 모델을 사용 하였는데, 이들은 서브 엘리먼트가 없는 엘리먼트들에 대한 필드의 반복을 지원하나, 서브 엘리먼트를 포함하고 있는 엘리먼트에 대한 필드의 반복을 지원하기 어렵다.(Sachs-Davis, 1994)

객체 기반 모델은 DTD의 엘리먼트를 클래스의 집합으로 변환하여 표현하는 방법으로서, DTD에 정의된 엘리먼트를 분석하고 이의 특성을 클래스의 정의에 반영하는 과정이 필요하다. 또한 recursive한 구조에 대한 질의를 제공하기 위해 질의를 확장해야 한다. 기존의 연구 중에 상당수가 객체 기반 모델을 따르고 있으며, 본 논문에서 제시한 방법 또한 이에 포함된다.

프랑스의 Francois는 SGML 문서에 대한 일반적인 저장소 데이터 모델을 제안하였는데, 이에 대한 그림은 다음과 같다.
(Francois, 1996)

그림에서 직사각형은 클래스를 의미하며, 클래스와 클래스사이에는 관계를 화살표로 표현하고 있다. 보통 화살표는 HAS-A를



〈그림 17〉 Francois의 SGML 저장소 데이터 모델

의미하며 원이 있는 화살표는 IS-A 관계를 의미한다. IS-A 관계를 가지는 하위 클래스는 상위 클래스로부터 모든 속성을 상속 받는다.

문서의 논리적인 구조를 표현하는 SGML 엘리먼트는 Empty 엘리먼트와 compound 엘리먼트를 서브 엘리먼트로 갖는다. Compound 엘리먼트는 여러개의 component들을 포함할 수 있으며, 따라서 비 단말노드라고 할 수 있다. 반면에 Empty 엘리먼트와 textual 엘리먼트는 단말노드라고 할 수 있으며, Textual 엘리먼트는 문서의 내용을 갖게 된다. SGML 엘리먼트는 SGML 애트리뷰트와 HAS-A 관계를 갖는다. SGML 애트리뷰트는 여러 개의 값들을 가질 수 있고, 엔티티를 참조할 수 있다. 엔티티는 매우 복잡한 구조를 가지고 있는데 크게 내부 엔티티와 외부 엔티티로 나눌 수 있다. 내부 엔티티는 문서의 어떤 부분이라도 올 수 있도록 component와 HAS-A관계를 갖는다. 외부 엔티티는 문서 외부에 존재하는 엔티티를 의미하는데, SGML 규칙을 따르며 외부 파일 형태로 존재하는 SGML 엔티티와 SGML 규칙을 따르지 않고 외부 파일 형태로 존재하는 DATA 엔티티로 나누어 모델링하고 있다.

본 논문에서 제시한 모델링과 비교를 해보면, Francois의 모델은 SGML 인스턴스에 중점을 둔 모델로 DTD가 포함하고 있는 제약 조건들에 대한 정보나 엔티티, Notation들에 대한 정보들을 포함하지 못한다는 단점을 갖는다.

독일의 GMD-IPSI에서는 다양한 DTD의 구조를 동적으로 데이터베이스에 반영할 수 있도록 하는 많은 연구를 하였다. GMD-IPSI에서는 분산 객체 지향 DBMS이며, 메타 클래스(metaclass)를 지원하는 VODAK 과 ASP SGML 파서를 확장 구현하였다. 그러

나 이는 특정 DBMS와 특정 SGML 파서를 확장하여 수행하였으므로, 이들에 의존적인 단점을 갖는다. 그리고, GMD-IPSI에서는 다양한 DTD를 동적으로 처리하기 위한 방법으로, DTD에 대한 정보 표현을 위한 수퍼 DTD(super-DTD)를 정의하고, 특정 DTD의 구조 정보를 이의 인스턴스로 표현하도록 하였다. 즉, 특정 DTD에 대한 구조를 데이터베이스에 반영하기 위해서는 먼저 이 DTD를 수퍼 DTD의 인스턴스 형태로 표현하고, 이 인스턴스를 이용해서 데이터베이스에 클래스들을 생성한다.(Aberer, 1994)

본 논문에서는 SGML 문서의 데이터 모델링을 DTD에 대한 정보을 유지하기 위한 모델 부분과 인스턴스에 대한 정보를 유지하는 부분으로 크게 나누어 모델링 하였으며, 특히 인스턴스에 대한 정보를 유지하는 부분은 특정 DTD의 구조를 분석하여 동적으로 생성되게 함으로서 SGML 인스턴스를 효과적으로 관리할 수 있도록 하였다.

6. 결론

본 논문에서는 SGML 문서의 효율적인 관리와 저장을 위해 SGML 문서의 논리적 구조를 객체 지향 방법으로 모델링하고, 이를 위한 SGML 문서 관리 시스템을 구현하였다. 본 논문에서 구현한 SGML 문서 관리 시스템은 스키마의 동적 생성 기능과 SGML 인스턴스 저장 기능 그리고 저장된 SGML 인스턴스의 문서 또는 문서 구조 단위로 꺼내는 기능을 제공하고 있다. 현재는 O2를 기반으로 구현하였으나, 본 논문에서 제시한 데이터모델은 어떠한 DBMS에도 쉽게 구현될 수 있다.

따라서, 본 논문은 대량의 SGML 문서를 처리하고 공유할 수 있는 기능을 제공하며, SGML 객체를 부분적으로 추출하거나 관리할 수 있는 기능을 제공하고 있다. 또한, 대량의 SGML 문서가 DBMS에서 제공하는 다양한 기능을 기반으로 관리될 수 있는 바탕을 마련하였다.

앞으로의 연구 방향으로는 엔티티 모델링에서 제외한 STARTTAG, ENDTAG, MS(marked section), MD(markup declaration)에 대한 엔티티들과 서브 문서 엔티티에 대한 모델링을 추가하는 것과 Hytime (Hypermedia/Time-based Structuring Language), DSSSL(Document Style Semantics and Specification Language), XML(eXtensible Markup Language) 등의 다양한 구조화된 문서들에 대한 데이터 모델링 방안에 대한 것이다.

참 고 문 헌

정희경, 현득창, 이수연. 1997. SGML 가이드. 사이버 출판사.

충남대학교. 1997. 구조화 정보 검색 색인 및 정보 집약 알고리즘 개발. SERI 위탁 과제 최종 보고서.

Aberer, K., et al. 1994. "The Prospects of Publishing Using Advanced Database Concepts." Conference on Electronic Publishing, Document Manipulation and Typography.

Christophides, V., Abiteboul, S., Cluet, S. and Scholl, M. 1994. "From Structured Documents to Novel Query Facilities." ACM SIGMOD'94.

Francois, P. 1996. "Generalized SGML repositories: Requirements and modeling." Computer Standards & Interfaces.

Herwijnen, Eric van. 1994. Practical SGML. Second Edition. Kluwer Academic Publishers.

International Organization for Standardization. 1986. Information Processing Text and Office System-Standard Generalized Markup Language (SGML). ISO/ IEC 8879:1986.

Lee, D.L. and Lee, W.C. 1994. "Using Path Information for Query Processing in Object-Oriented Database Systems." Third International Conference on Information and Knowledge Management.

Lee, Kyuchul, Lee, Y. K., Yoo, S. J., Yoon, K. and Berra, P. B. 1996. "Object-Oriented Modeling, Querying, and Indexing for Multi-Structured Hypermedia Document Database." Proc. of the IEEE Int. Workshop on Multi-Media Database Management Systems '96.

Maler, Eve and Andaloussi, Jeanne El. 1996. Developing SGML DTDs. Prentice Hall.

O2 Technology. O2 Engine API Reference Manual. O2 Technology.

O2 Technology. OQL User Manual. O2 Technology.

Sachs-Davis, R., Arnold-Moore, T. and Zobel, J. 1994. "Database Systems for Structured Documents." Informational Symposium on Advanced Database Technologies and Their Integration.