DTD 의존 스키마에 기반한 SGML 문서 저장 시스템 개발에 관한 연구

김 현 기† · 노 대 식† · 강 현 규††

요 약

차세대 정보 포맷 표준으로 부상되고 있는 SGML(Standard Generalized Markup Language) 문서들을 데이터베이스에 저장하여 관리하기 위한 전자 문서 관리 시스템에 대한 요구가 다양한 응용분야에서 급증하고 있다. 본 논문에서는 계층적 트리 구조를 갖는 SGML 문서를 정보 손실 없이 데이터베이스 시스템에 저장하기 위해 고려해야 할 사항들을 기술하며, ODMG(Object Database Management Group) 객체 모델을 사용하여 SGML 문서를 저장하기 위한 데이터 모델을 제안한다. 제안하는 모델은 SGML 문서의 물리적 엘리먼트 구조와 논리적 엔티티 구조를 반영하여 엘리먼트 단위로 정보 손실 없이 SGML 문서를 데이터베이스 시스템에 저장 할 수 있으며, ODMG 호환 가능한 객체 지향 데이터베이스 시스템들에 쉽게 이식되어 사용 될 수 있다. 그리고 제안한 데이터 모델에 따라 개발된 SGML 문서 저장 시스템의 구현에 대해 설명한다. 구현한 SGML 문서 저장시스템은 임의의 DTD(Document Type Definition)에 대해 동적 데이터베이스의 스키마 생성 기능, SGML 문서 저장 기능, 저장된 데이터베이스 객체로부터 SGML 문서로의 복원 기능과 저장된 데이터베이스 객체들에 대해 색인/검색 기능 등을 수행한다.

A Study on Development of SGML Repository System Based on DTD-dependent Schema

Hyun-Ki Kim† · Dae-Sik Roh† · Hyun-Kyu Kang††

ABSTRACT

In various fields of information technology, it is growing up the needs about dynamic content management systems to store and manage SGML(Standard Generalized Markup Language) documents in a database system. In this paper, we consider the issue of storing SGML documents that having complex hierarchical structure into a database system, and then propose a data model based on ODMG(Object Database Management Group) object model in order to store SGML documents without loss of information. Because the proposed data model reflects physical element structure and logical entity structure of SGML documents, it is able to store the SGML document in a database system at the element- level granularity without any information loss. And also the proposed data model can be adapted among ODMG-compliant object database management systems. Finally, we will discuss on the implementation details of SGML repository system according to our data model. Our SGML repository system supports the functionality of automatic database schema creation for any DTD(Document Type Definition), the functionality of storing the SGML document, the functionality of dynamic document assembly from stored database objects to SGML document, and the functionality of indexing and searching for database objects.

† 정 회 원 : 한국전자통신연구원 연구원
†† 종신회원 : 한국전자통신연구원 선임연구원
논문접수 : 1998년 9월 21일, 심사완료 : 1999년 3월 11일

1. 서 론

종이 문서에 기반한 정보의 생성, 공유, 보관, 교환은 대규모의 정보가 빈번히 갱신되는 동적인 환경에서는 관리가 불가능하므로, 전자 정보에 의한 문서 관리의 필요성이 요구되고 있다. 문서 정보 구조화 언어인 SGML(Standard Generalized Markup Language)[1]은 1986년에 국제표준(ISO 8879)으로 제정된 후 전자 정보의 저장과 교환을 위한 표준으로 채택되고 있다. 최근 W3C(World Wide Web Consortium)에서는 SGML의 복잡한 기능들을 단순화시킨 XML(eXtensible Markup Language)을 1998년 2월에 권고안(recommendation)으로 제정하여 SGML/XML에 기반한 웹 출판, CALS/EC(Commerce At Light Speed/Electronic Commerce), 디지털 도서관(digital library), EDI(Electronic Data Interchange), 기술 문서 관리 시스템 등의 다양한 응용 분야에서 사용될 전망이다[2]. XML은 SGML 문서를 웹 환경에서 쉽게 저작, 관리, 전송, 공유하기 위해 SGML의 복잡한 특성들을 제거한 SGML의 부분 집합이다[2].

SGML이 이처럼 광범위한 응용분야에서 채택되는 이유는 다음과 같다. 첫째, 정보 생성에 드는 비용과 시간을 줄일 수 있기 때문이다. 기존 워드 프로세서나 데스크 탑 출판 시스템을 이용한 정보의 생성시 저자는 30%의 시간을 정보를 찾는데 소비하며, 30%는 글자의 폰트 종류, 크기 및 페이지 레이아웃 등 보기 좋은 문서를 출력하는 작업에 소비한다. 또한 매 18개월마다 소프트웨어 기술이 변하므로 기존에 생성한 데이터 변환에 비용을 투자해야 한다[3]. SGML/XML은 다수의 응용 프로그램에 의해 생성된 다양한 유형의 멀티미디어 데이터로 구성된 가상 문서(virtual document) 개념을 지원하기 때문에 기존에 생성된 정보의 단위들을 조합하여 새로운 정보를 쉽게 생성 할 수 있다. 즉, SGML은 정보를 효율적으로 처리하기 위해서 정보를 비트나 바이트의 스트림으로 처리하지 않고 "하나의 단위로 참조될 수 있는 문자들의 집합"인 엔티티(entity)를 정의한다[1]. 이와 같은 엔티티 기능은 동일한 정보를 다양한 목적에 따라 효율적으로 재구성, 재사용 할 수 있게 하며, 데이터베이스 시스템을 저장소로 이용하여 사용자 요구에 의한 문서 인스턴스를 동적으로 생성 할 수 있도록 한다[4,5,6,7]. 둘째, 시스템에 독립적인 문서 교환 및 처리가 가능하기 때문에 이질적인 네트워크 및 분산 환경하에서의 전자 상거래 및 EDI의 정보 기술 언어로 활용 할 수 있다. 셋째, 문서의 논리적인 마크업(markup)과 포맷팅 정보를 분리시킴으로써 특정 문서 집합에 대한 표준화된 출력과 동일 문서에 대한 다수의 출력 포맷을 지정 할 수 있다. 이는 문서의 내용을 작성하는 저작 과정에서 문서의 최종 출력을 위한 포맷 정보를 작성하는 비용을 완전히 제거 시키는 효과를 가져올 뿐 아니라, DSSSL(Document Style Semantics and Specification Language)과 XSL(eXtensible Style Language)의 다양한 스타일시트를 이용한 개인화된 출판을 가능하게 한다. 마지막으로 SGML 문서는 DTD에서 정의된 논리적인 구조에 따라 문서가 작성되므로 검색의 범위를 특정 엘리먼트, 엔티티, 속성 등으로 한정한 질의가 가능함으로써 검색 결과를 향상시킬 수 있다[4,5,6,7].

그러나 이와 같은 SGML 문서의 생명주기 전 과정에서 정보의 생산성, 재사용성, 지속성, 이식성 등과 같은 SGML 사용의 장점들을 얻기 위해서는 SGML 엔티티들을 저장하고 관리 할 수 있는 문서 관리 시스템이 필수적으로 요구된다.

SGML 문서를 관리하기 위한 저장 시스템의 구현시 고려해야 될 사항들은 다음과 같다.

① SGML 문서 관리 개념은 SGML 문서를 공동 편집하고자 하는 요구로부터 발생된다. SGML 저장 시스템은 워크그룹 환경에서 다수 저자가 동일한 문서의 일부분에 대한 편집, 갱신 등을 지원하기 위한 공동저작 기능을 제공해야 한다. 그러므로 사용자는 "document fragment server"를 필요로 하며 교환과 관리의 단위는 문서가 아니라 프레그먼트(fragment)이다[7].

② SGML DTD는 문서의 클래스를 정의하며 문서는 클래스의 인스턴스로 맵핑 될 수 있으므로 SGML 문서를 컴포넌트별로 나누어 데이터베이스에 저장하고 원래 문서로 복원하기 위한 모델링이 필요하다. SGML 문서를 저장하기 위한 데이터 모델은 저장 시스템에 삽입하기 전의 문서와 저장된 문서를 다시 복원했을 때의 데이터 손실을 방지하기 위해 SGML의 모든 특성들을 고려해야 한다.

③ 저장소에 저장된 SGML 문서의 프레그먼트를 사용자 요구 또는 질의에 의해 원래의 SGML 문서로 재통합하는 동적인 문서 어셈블링(dynamic docu-

ment assembling) 기능이 제공돼야 한다.

④ 저장소에 저장된 SGML 문서의 구조, 속성, 키워드에 대한 색인과 검색 기능이 제공 되어야 한다. 사용자는 정보 검색시 검색 범위를 문서의 특정 부분으로 한정시켜 질의를 할 수 있으므로 보다 정확한 검색 결과를 얻을 수 있다. 또한 데이터베이스 질의는 SGML 문서 구조에 대한 정확한 경로 표현(path expression)을 지정해야 하나, 정보 검색은 문서의 정확한 구조를 모르더라도 구조에 대한 검색을 할 수 있다.

본 논문에서는 대량의 SGML 문서를 데이터베이스 시스템에 저장하여 관리하기 위해 필요한 SGML 데이터 모델을 제안하며, 제안한 모델에 따라 SGML 문서를 엘리먼트 단위로 나누어서 저장하고, 데이터베이스에 저장된 객체들을 참조하여 동적으로 문서를 어셈블링하는 방법을 제안한다. 또한 저장소에 프래그먼트로 저장된 데이터베이스 객체에 대해 색인 및 검색 할 수 있는 방법을 기술한다.

논문의 구성은 다음과 같다. 2장에서는 SGML 문서 관리 시스템에 대한 관련연구에 대해 기술하며 3장에서는 SGML 데이터 모델에 대해 기술한다. 4장에서는 객체 지향 SGML 저장 시스템의 설계 및 구현에 관해 기술한다. 마지막 5장에서는 결론 및 향후 연구 방향을 제시한다.

2. 관련연구

SGML 문서를 저장하여 관리하기 위한 기존의 연구는 관계형 모델(relational model), 확장 관계형 모델(extended relational model), 객체 기반 모델(object-based model) 등으로 구분된다[6]. 최근의 연구들은 대부분 확장 관계형 모델과 객체 기반 모델을 사용하므로 이들 모델들에 대한 연구동향들을 상세하게 기술한다.

관계형 모델을 이용한 방법은 엘리먼트를 관계형 데이터베이스 시스템에 테이블로 저장하고 SQL를 이용하여 SGML 문서 또는 문서의 특정 필드를 검색하는 방법이다. 그러나 관계형 모델을 이용한 방법은 DTD가 복잡할수록 엘리먼트를 저장하기 위한 테이블의 수가 기하급수적으로 늘어나고 SQL를 이용한 검색시에도 다수의 테이블에 산재해 있는 레코드들에 대해 고비용의 죠인(join) 연산이 요구되는 단점이 있다[6].

이에 반해 확장 관계형 모델과 객체 기반 모델은 트리 구조의 관계성이 데이터 모델에 반영되므로 죠인 연산이 불필요하며, 상속 및 추상화를 통해 SGML을 자연스럽게 모델링 할 수 있다[6,7]. 확장 관계형 모델에 기반한 최초의 상용 SGML 문서 관리 시스템인 TEXCEL International의 Information Manager[8]는 객체 관계형 데이터베이스 시스템인 UniSQL를 하부 저장 시스템으로 사용한다. Information Manager의 주요 기능들로써는 엘리먼트 단위로 나누어서 문서 저장, 공동 저작을 위한 check-in/check-out 기능, 버전 제어, SQL를 이용한 검색 기능 등이 있다. 그러나 Information Manager(버전 1.0)는 "IGNORE" 참조구간(Marked Section)을 지원하지 못하므로 "IGNORE" 참조구간이 포함된 문서들을 저장한 후 원래의 SGML 문서로 조합하는 경우에는 정보를 손실하며, 또한 외부 엔티티 파일들이 포함된 문서를 저장하는 경우에는 사용자가 모든 외부 엔티티 파일들을 직접 등록시켜야 하는 번거로움이 있다[8].

Patricia Francois[7], 한에노[9] 등은 객체 기반 모델을 사용하여 SGML 문서를 객체 지향 데이터베이스 시스템에 저장하여 관리하기 위한 데이터베이스를 설계하였다. 위의 두 접근 방법에서는 전자문서의 논리적 구조를 기술하는 SGML DTD 정보를 관리하기 위한 객체지향 메타 스키마를 제안하고, 이 메타 스키마로부터 동적으로 생성된 응용 데이터베이스 스키마를 통해 전자 문서를 객체 지향 데이터베이스에서 관리하는 방법을 제안한다. 그러나 제안된 방법은 다음과 같은 단점을 갖는다. 첫째, 엘리먼트의 선언내용인 내용 모델(content model)이 한 개의 구성요소로 이루어져 있는 SINGLE_MODEL과 여러 개의 구성요소로 이루어져 있는 GROUP_MODEL로 구분되므로, GROUP_MODEL로 선언된 내용모델인 경우에는 GROUP_MODEL를 추상화시킨 임시 노드가 엘리먼트와 엘리먼트 사이에 위치한다[7,9]. 그러므로 실제 SGML 문서가 파싱되어 저장되는 경우에 파스 트리에 존재하지 않는 노드가 데이터베이스에 생성되어 저장되므로 저장 시간의 증가와 저장 공간의 낭비를 초래 할 수 있으며, 사용자가 문서 구조에 대한 검색시 객체의 경로를 표현하기가 어렵다.

객체 기반 모델에 근거한 또 다른 접근 방법으로 프랑스의 INRIA(The French National Institute for Research in Computer Science and Control)에서는 이

질적인 분산환경하에서 대량의 전자문서들에 대해 문서관리, 데이터 재구성, 질의 최적화 등을 수행하는 VERSO 프로젝트를 수행하고 있다. VERSO 프로젝트의 주요 접근 방법은 객체 지향 데이터베이스 시스템인 O2를 이용하여 SGML DTD는 O2의 스키마로 맵핑시키며, SGML 문서는 생성된 스키마의 객체와 값으로 데이터베이스에 맵핑시키는 것이다. 그리고 데이터베이스에 저장된 구조화된 문서에 대해 검색하기 위해 O2의 질의 언어인 O2SQL를 확장시키는 접근방법을 사용하고 있다[4]. 그러나 이와 같은 접근 방법은 다음과 같은 문제점들을 갖고 있다. 첫째, SGML DTD를 데이터베이스의 스키마로 변환하는 과정을 시스템이 처리하는 것이 아니라 사람이 수동으로 변환해야 한다. 둘째, 문서를 저장한 후 원래의 SGML 문서로 복원할 때 O2SQL를 사용하므로 검색 결과가 SGML 문서가 아닌 O2 객체인 단점을 갖고 있다. 또한 대량의 문서 컬렉션에 대해서는 검색 성능이 떨어진다.

본 논문에서는 기존 연구에서 SGML DTD의 데이터베이스 스키마로의 자동변환시 엘리먼트의 내용 모델이 트리상의 노드로 나타나는 문제, SGML 문서 저장시 "IGNORE" 참조 구간, 자식 노드의 순서 및 위치 관계 보존, 문서 내에서 참조된 외부 엔티티를 객체 참조로 변환하는 문제 등을 해결하는 데이터 모델을 제안한다. 이외에도 SGML에서 정의된 엘리먼트, 엔티티, 속성, PI(Processing Instruction), 주석, 참조 구간 등 다양한 특성들을 데이터베이스 스키마에 반영하기 위한 데이터 모델을 제안한다.

3. SGML 데이터 모델

3.1 SGML 기본 개념

SGML은 임의의 응용 분야에 대해서도 사용 될 수 있는 문서 형을 정의하는 메타 언어(meta-language)로서 SGML 선언부(declaration), 문서형 정의부(DTD) 및 문서 인스턴스(document instance)의 세 부분으로 구성된다[1]. SGML 선언부는 문서에서 사용되는 구체 구문(concrete syntax), 마크업과 문서내에서 사용되는 문자 집합, SGML 특성들(features)을 정의한다. 문자 코드 체계가 다른 이기종 시스템에서 SGML 문서를 전송 받더라도 선언부의 정보를 이용한 처리(예 : 문자 코드 변환)를 통해 원본 문서의 의미를 손실하지 않고 문서를 처리할 수 있다. DTD는 엘리먼트 선언, 속성 정의 리스트 선언, 엔티티 선언 등을 포함하며, 엘리먼트 선언은 부엘리먼트들의 순서와 발생 횟수를 나타낸다. 문서 인스턴스에는 DTD에서 정의된 구조에 따라 작성된 데이터와 마크업을 포함한다. 다음의 (그림 1)은 메모 DTD를 나타내고 있다.

```
<!ENTITY   logo      SYSTEM /usr/hkkim/logo.bmp NDATA BMP >- ❶
<!ENTITY   맺음말    <맺음말>감사합니다.</맺음말>          >
<!ENTITY   % doctype 메모                                 >
<!NOTATION bmp  PUBLIC "+//ISBN 0-7923-9432-1::Graphic
           Notation//NOTATION Microsoft Windows bitmap//EN">
<!ELEMENT  %doctype; ((수신 & 발신), 본문, 맺음말?)       > - ❷
<!ELEMENT  수신      (#PCDATA)                            >
<!ELEMENT  발신      (#PCDATA)                            >
<!ELEMENT  본문      (제목, 단락)+                        > - ❸
<!ELEMENT  제목      (#PCDATA)                            >
<!ELEMENT  단락      (#PCDATA | 그림 | 단락)*             > - ❹
<!ELEMENT  그림      EMPTY                                >
<!ELEMENT  맺음말    (#PCDATA)                            >
<!ATTLIST  그림      파일    ENTITY    #IMPLIED           > - ❺
```

(그림 1) 예제 memo DTD

다음의 <표 1>과 <표 2>는 DTD 작성에 사용되는 발생 지시자(occurrence indicators)와 연결자(connectors)의 의미를 나타내고 있다. (그림 1)의 DTD에서 알 수 있듯이 SGML 엘리먼트는 계층적 트리 구조를 가지며 트리상의 엘리먼트는 발생 지시자와 연결자에 의해 정의된 내용 모델(content model)에 따른 문맥에 따라 유효하게 작성돼야 한다. (그림 1)의 ❶은 외부 엔티티를 정의하고 있으며, ❷의 내용 모델에서는 "수신"과 "발신" 엘리먼트가 순서와 무관하게 "메모" 엘리먼트의 부엘리먼트(subelement)로 올 수 있으며, "맺음말"은 생략 가능한 내용 모델을 기술하고 있다. ❸과 ❹의 내용 모델에서는 발생 지시자의 의미에 따라 1번 이상 "(제목, 단락)"이, 0번 이상 "(#PCDATA, 그림, 단락)"이 발생돼야 함을 의미한다. 또한 ❹에서는 "단락" 엘리먼트의 부엘리먼트가 "OR"의 관계로 나타내는데, "단락" 엘리먼트는 트리상에서 재귀적으로 나타날 수 있다.

〈표 1〉 발생 지시자

기호	의 미
?	발생될 수도 있고, 발생 안될 수도 있음
+	1번 이상 반복하여 발생함
*	0번 이상 반복하여 발생함

〈표 2〉 접속자

기호	의 미
|	OR 접속자
&	AND 접속자
,	Sequence 접속자

3.2 SGML 데이터 모델의 고려사항

본 절에서는 위의 사항들을 고려하여 SGML 데이터 모델과 DTD에 정의된 엘리먼트의 구조에 따른 ODBMS의 스키마를 동적으로 생성하는 기법에 대해 설명한다. SGML 엘리먼트를 ODBMS의 클래스로 직접 사상하여 저장할 때 해결해야 될 구체적인 문제점들은 다음과 같다.

- SGML 문서는 트리로 구성되며, 특정 노드의 자식들은 서로 다른 유형의 엘리먼트가 올 수 있으므로 자식 노드들의 순서 관계가 유지되어야 하나 ODBMS의 클래스로 직접 사상 시킬 경우에는 순서 관계를 유지시킬 수 없으므로 원래의 문서로 복원시킬 경우 순서 관계가 손실 될 수 있다[4, 6,7].
- SGML의 발생 지시자와 연결자 의미를 표현 할 수 있는 연산자들을 ODBMS가 지원하지 못한다. (그림 1)의 ❸에서 "(제목, 단락)+"는 반드시 1번 이상 인스턴스가 발생되야 하므로 객체의 인스턴스가 nil 값이 할당되지 않도록 제약조건이 설정되어야 하며, ❹의 "(#PCDATA | 그림 | 단락)*"는 OR의 관계로 0번 이상 인스턴스가 발생되어야 하므로 "단락" 클래스는 "#PCDATA", "그림", "단락" 클래스를 "OR" 접속자의 의미를 갖는 Union 타입의 튜플로 가져야 한다. 그러나 대부분의 ODBMS는 Union 타입을 지원하지 못한다[4,7].
- SGML에서는 DTD 또는 문서 형 선언 부분 집합(Document Type Declaration Subset)에서 엔티티를 선언하고 문서에서 엔티티 참조를 통하여 정보를 공유하는 방법을 제공한다. 다음 (그림 2)의 문서에서는 문서 형 선언 부분 집합을 포함하고 있으며, ❷와 ❹에 나타나듯이 SGML 문서에는 엔티티 참조가 발생하므로 엔티티 참조를 저장할 때에는 객체 참조로 변환 할 수 있어야 한다.

```
<!DOCTYPE 메모 SYSTEM "메모.dtd" [                                    ❶
<!ENTITY figa SYSTEM "jade.bmp" NDATA bmp>
]>
<메모>
<수신>신영미</수신>
<발신>김현기</발신>
<본문><제목>Jade (General SGML/XML tool)</제목>
<단락>본 그림은 Jade의 시스템 구성도입니다. <그림 파일=figa>          ❷
<단락>Jade는 James Clark이 구현한 DSSSL엔진입니다.</단락></단락></본문>❸
&맺음말;                                                              ❹
</메모>
```

(그림 2) 예제 문서("메모.sgm")

3.3 SGML 데이터 모델

SGML데이터 모델은 ISO 8879[1]에 명시된 SGML 문법에 기반한 모델로서 특정 DTD에 무관하게 어떠한 DTD에 따른 문서 인스턴스라도 공통적으로 동일한 구조를 갖는 스키마를 말한다. 특정 DTD스키마를 생성하기 위한 하부 스키마의 역할을 하는 메타 스키마로 지칭할 수 있다. SGML 데이터 모델은 ODMG 객체 모델인 C++ ODL(Object Definition Language)의 persistence-capable 클래스와 컬렉션 클래스들을 사용하여 구현하였다[10,11].

3.3.1 엘리먼트 모델(Element Model)

엘리먼트 모델은 엘리먼트의 트리상의 위치 정보, 속성 정보, 자식 노드의 순서 관계 등을 포함하여야 한다. SGML표준에 정의된 엘리먼트 구문 모델은 엘리먼트의 종류에 따라 크게 단말 엘리먼트(Terminal Element)와 비단말 엘리먼트(Non-Terminal Element)로 구분될 수 있다. 이와 같이 SGML의 엘리먼트 구문 모델에 근거하여 (그림 3)과 같이 엘리먼트 모델을 설계하였다.

(1) 엘리먼트 클래스(Element Class)

엘리먼트 클래스는 단말 노드 또는 비단말 노드의 구분없이 모든 엘리먼트가 갖는 공통적인 속성을 추상화하여 만든 슈퍼클래스가 "Element"로서 문서 트리에서의 위치 정보, 항해 연산을 갖는다. (그림 3)에서 보는 바와 같이 엔티티 참조부(entity reference)도 엘리먼트로부터 상속을 받는 이유는 엔티티 참조부가 임의의 문서 트리 위치에 삽입될 수 있을 뿐만 아니라 엔티티 참조부가 가리키는 엔티티 자체도 엘리먼트로 구성된 서브 트리를 가질 수 있기 때문에 기본적으로 트

리의 노드와 같은 기능을 갖는다.

(2) NT 엘리먼트 클래스(Non-Terminal Element Class)

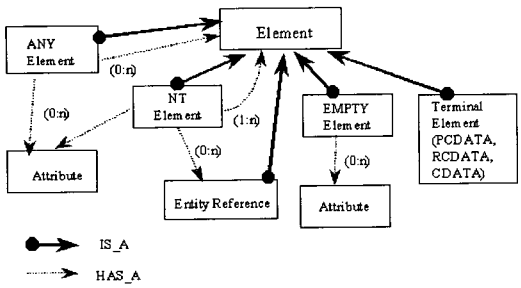
NT 엘리먼트 클래스는 내용모델을 갖는 엘리먼트로서, 자신의 자식 엘리먼트를 한 개 이상 갖는다. 이 클래스는 자신의 구성 요소가 되는 자식 엘리먼트들이 어떤 엘리먼트들로 구성되는지, 어떤 순서로 배열되어 있는지 등에 대한 정보를 갖게 된다. 즉, 내용 모델에 명시된 엘리먼트의 구성 요건에 따라 스키마가 생성되어야 하므로 DTD스키마 생성시에 내용 모델을 갖는 대부분의 엘리먼트들이 이 클래스로부터 상속을 받게 된다.

(3) 단말 엘리먼트 클래스(Terminal Element Class)

PCDATA, CDATA, RCDATA는 실제 문서 텍스트만을 스트림으로 가지며, 자신의 자식 노드를 갖지 않는 트리상의 단말노드이다. 따라서 단말 엘리먼트에 속하는 클래스들은 특정 DTD에 따른 스키마 생성시 베이스 클래스로 이용되지 않고 인스턴스 객체 트리에 임포트되어 사용된다.

(4) ANY, EMPTY 엘리먼트 클래스(ANY, EMPTY Element Class)

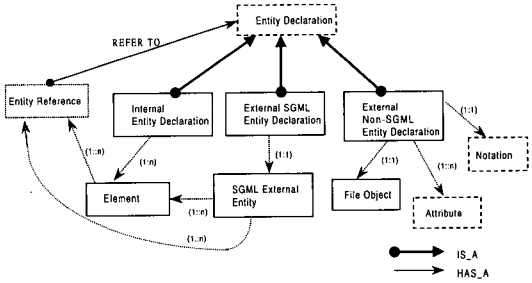
ANY, EMPTY 로 선언된 엘리먼트는 비단말 또는 단말 엘리먼트와는 확연히 다른 특성을 가지므로 각각 별개의 클래스로 정의하였다. ANY는 어떠한 엘리먼트든지 자식 노드를 가질 수 있고 자신의 어트리뷰트를 가질수 있으며, EMPTY는 자식 노드는 없으나 반드시 어트리뷰트를 갖는다.



(그림 3) 엘리먼트 모델

3.3.2 엔티티 모델(Entity Model)

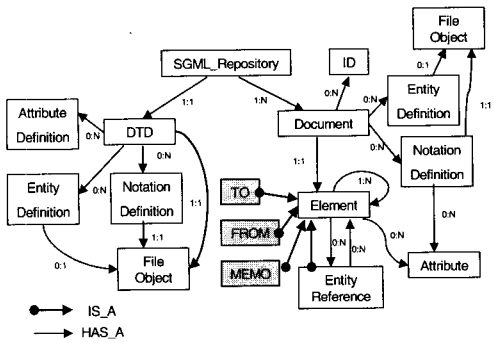
SGML에서는 DTD나 문서형 선언 부분 집합에서 엔티티를 선언하고 문서에서는 엔티티 참조를 통하여 선언된 엔티티를 공유하여 사용할 수 있다[1,2]. 그러므로 엔티티 선언은 유형에 따라 내부 엔티티(internal entity), 외부 텍스트 엔티티(external text entity), 외부 데이터 엔티티(external data entity), 외부 파라미터 엔티티(external parameter entity)의 세부 클래스로 파생되며, DTD 또는 문서형 선언 부분집합에서 저장하고 문서의 파스 트리에 나타난 엔티티 참조는 실제 데이터가 저장되어 있는 엔티티 선언 클래스를 가르키게 하여 동일 정보를 중복하여 저장하지 않도록 한다. 다음 (그림 4)는 엔티티 모델을 나타내고 있다.



(그림 4) 엔티티 모델

3.3.3 저장소를 위한 전체 모델

SGML 모델은 DTD, 문서 인스턴스, 외부 엔티티, 외부 엔티티의 데이터를 모델링하기 위한 SGML 클래스들과 이 클래스들을 가상 디렉토리 구조로 관리하기 위한 시스템 클래스들로 구성된다. 아래의 (그림 5)는 SGML 문서관리를 위한 저장소의 전체 모델을 나타내고 있다. 여기서 특정 DTD스키마의 예로 "memo", "to", "from"클래스가 엘리먼트 클래스의 상속을 받는 것으



(그림 5) SGML데이터 모델

로 나타나 있는데, 좀더 구체적으로는 비단말 클래스들인 NT, EMPTY, ANY중에서 상속받게 된다. 즉 DTD상의 엘리먼트들은 데이터를 가지는 단말 클래스(PCDATA, RCDATA, CDATA)를 자식 노드로 갖기 때문이다. 이에 대한 자세한 설명은 4장에서 한다.

■ SGML Repository클래스

SGML Repository 클래스는 4장에서 설명되는 스키마 생성기에 의해 DTD당 하나의 오브젝트가 생성되며, DTD에 대한 정보를 담는 틀인 DTD클래스와 이 DTD에 따라 작성된 다수의 문서를 저장할 수 있도록 한 Document 클래스의 리스트로 구성된다.

■ DTD 클래스

DTD 클래스는 다수의 속성 정의 리스트 선언(Attribute Definition List Declarations), 엔티티 선언(Entity Definition), 표기법 선언(Notation Definition)과 DTD화일 자체의 저장을 위한 File Object를 갖는다. 시스템에 저장되는 엔티티 선언 정보는 DTD에 정의된 내부 또는 외부 엔티티를 비롯하여 파라미터 엔티티까지 포함하며 외부 엔티티의 파일 정보는 File Object를 통해 접근 가능하다.

■ Document클래스

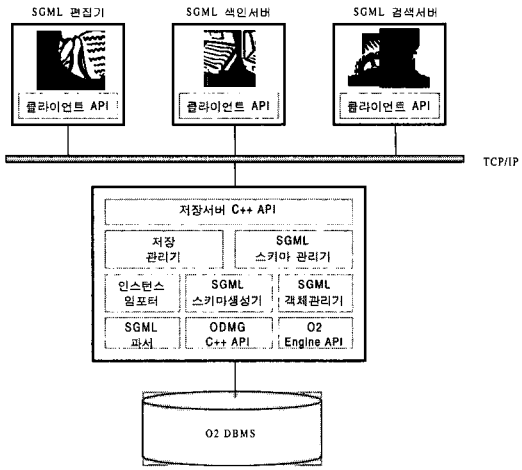
Document 클래스는 루트 엘리먼트를 가리키는 지속성 객체를 속성으로 가지며, 문서 자체 내에 추가로 정의된 각종 선언부를 저장하기 위해 엔티티, 표기법 선언 등을 가지며, 이들 정보들은 엔티티 참조를 통해 문서에서 사용되며 문서를 데이터베이스 외부로 익스포트할 때 선언부를 그대로 복원할 수 있게 한다.

4. 객체 지향 SGML 문서 저장 시스템 구현

본 장에서는 3장의 데이터 모델에 근거한 SGML 문서의 저장, 복원, 구조 정보 검색 기능 등을 지원하는 SGML 저장시스템에 대해 기술한다. 4.1절에서는 시스템의 구조에 대해 설명하며, 4.2절에서는 각 모듈의 기능에 대해 설명한다. 4.3절에서는 색인기 및 검색기와의 인터페이스에 대해 기술하며, 4.4절에서는 구현 환경에 대해 설명한다.

4.1 시스템 구조

구현한 SGML 문서 저장 시스템의 하부 저장시스템으로는 O2 ODBMS을 사용하였으며 TCP/IP(Transmission Control Protocol/Internet Protocol)를 이용한 연결 지향형 클라이언트-서버 구조로 구현하였다. O2는 페이지 서버 아키텍쳐를 갖는 객체 지향 데이터베이스 시스템으로 ODMG-93에 따른 C++ 바인딩과 OQL(Object Query Language)를 지원하며 상속과 다형성 등의 객체 지향 개념 등을 제공한다[10,11]. 전체 시스템의 구조는 아래의 (그림 6)에 나타나 있다.



(그림 6) SGML 저장시스템 구조

시스템 구성 요소들로는 ODMG C++로 구현한 저장 관리기, SGML 스키마 관리기, SGML 객체 관리기와 SGML 스키마 생성기, 인스턴스 임포터와 KLESP 파서로 구성되며, 클라이언트 응용 프로그램 개발을 위한 라이브러리가 있다. 본 시스템의 주요 기능들은 응용 프로그램에 대한 세션 개방 및 종결, 동적 스키마 생성, 인스턴스 저장 및 삭제, DTD 구조 정보 제공, 저장된 문서의 색인 포맷 전달을 색인기에 전달, 문서 또는 일부분의 OID에 의한 검색 기능 등이다. 또한 저장시스템의 클라이언트 응용 시스템으로 본 연구실에서 개발한 SGML 편집기와 구조 정보 색인기/검색기를 본 시스템과 연동되어 사용할 수 있도록 확장 중이다.

4.2 시스템 구성 모듈

4.2.1 저장 관리기

저장관리기는 외부 응용 프로그램의 요구에 따라 세션을 개방하거나 종료하며 각 모듈에 대한 프로세스

제어 및 메시지 분배의 역할을 한다. 클라이언트-서버 환경의 다사용자 접근을 지원하기 위해 저장 관리기 내부 모듈은 프로세스 그룹을 구분하기 위해 세션의 PID(process ID)를 사용한다. 사용자는 저장소(Repository)의 ID를 사용하여 저장 관리기에 접근함으로써 O2 DBMS의 스키마와 베이스로부터 추상화된 뷰를 가지며 SGML DTD는 저장소 ID와 1대 1로 대응된다. 즉, 한 개의 저장소에는 하나의 DTD 스키마와 베이스가 존재하게 된다.

4.2.2 SGML 파서

SGML 파서의 출력인 ESIS(Element Structure Information Set)는 문서가 갖는 모든 정보를 포함하고 있지 않으므로 이를 이용한 문서 저장은 정보의 손실을 초래한다. 예를 들어 "IGNORE"로 선언된 마크구간의 데이터는 파서가 파싱을 하지 않기 때문에 ESIS을 이용하여 문서를 저장하는 경우에는 데이터의 손실을 유발한다. 또한 인스턴스에 참조되지 않은 엔티티, 표기법 등도 ESIS에는 정보가 누락되어 있다. 따라서 저장시스템에 사용되는 파서의 출력물은 DTD에 선언된 엔티티, 표기법 등의 정보와 "IGNORE"로 선언된 마크구간의 정보를 모두 제공 할 수 있어야 한다.

저장시스템에서 사용한 파서는 본 연구팀에서 개발한 KLESP를 사용하였으며 KLESP는 DTD와 인스턴의 파싱결과로 객체 트리로 구성된 메모리 포맷을 제공한다. DTD 파싱 정보는 RT(Rule Tree)로 제공되며, 인스턴스 파싱 결과는 OXF(OX Format)으로 제공된다. OXF 트리는 인스턴스의 엘리먼트가 노드로 구성된 트리로써 DTD에서 정의된 내용모델과 속성 정의에 대한 정보를 참조하기 위해 RT에서 해당 엘리먼트가 정의된 노드를 참조한다.

4.2.3 SGML 스키마 관리기

본 시스템에서는 DTD별로 O2에 새로운 스키마와 베이스를 생성한다. 사용자는 DTD별로 유일한 SGML 저장소 이름을 명명하여 문서 인스턴스를 저장하며, 문서를 저장하는 경우에도 유일한 문서의 이름을 부여하도록 하였다. 이 경우 사용자는 데이터베이스에 대한 지식이 없어도 저장소 또는 문서 이름을 사용하여 데이터를 접근 할 수 있다.

SGML 스키마 관리기는 저장소 이름과 O2의 스키마와 베이스 이름, 사용자 정의 문서 이름과 논리적 객체 식별자에 대한 카탈로그 서비스와 저장된 문서의 색인 여부에 대한 정보를 O2의 스키마로 정의하여 저장 관리기의 요청에 따른 맵핑 서비스를 제공한다. SGML 스키마 관리기는 저장관리기와 별도의 프로세스로 수행되며, 프로세스간의 통신은 UNIX 시스템의 IPC(Inter-Process Communication) 설비인 메시지 큐와 공유 메모리를 이용하여 통신한다.

SGML 스키마 관리기의 역할은 다음과 같다.

- 저장소 이름과 사용자 정의 문서 이름의 유일성 검사 : 스키마 생성과 인스턴스 저장시 동일 이름이 이미 등록되어 있는지 검사함.
- 저장소 이름 등록 : 스키마 생성의 트랜잭션이 commit된 경우, 저장소 이름과 O2의 스키마, 베이스 이름의 등록
- 사용자 정의 문서 이름 등록 : 인스턴스 저장의 트랜잭션이 commit된 경우, 사용자 문서 이름과 객체 식별자 등록
- 저장소 이름에 대한 O2의 스키마, 베이스 이름 맵핑
- 색인기에 색인되지 않은 문서의 식별자 반환
- 색인된 문서에 대한 색인 정보 갱신

4.2.4 SGML 스키마 생성기

스키마 생성기는 ODMG C++로 코딩되어 이미 데이터베이스에 등록된 SGML 스키마를 임포트하여 DTD에 따른 스키마를 자동으로 생성하는 역할과 DTD에 선언된 엔티티, 속성, 표기법에 대한 항목들을 저장하는 역할을 수행한다. 즉, 스키마 생성기는 저장 관리기로부터 DTD 파일, 스키마 이름, 데이터베이스 이름 등을 입력인자로 받아 해당 DTD에 따라 작성된 문서를 저장할 SGML 저장소의 스키마 생성 및 DTD에 포함된 엔티티, 속성, 표기법 정의 정보들을 저장한다.

SGML 저장소의 생성은 다음의 3단계로 이루어진다.

첫째, 전처리 단계로서 입력된 DTD화일을 파싱한 후, 그 결과물인 RT(Rule Tree)를 이용하여 위해 각 엘리먼트마다 해당 내용 모델에 근거한 자식 엘리먼트의 구성요소를 추출하고, 엘리먼트의 이름(generic identifier)이 한글인 경우에는 클래스 이름으로 한글을 사용할 수 없으므로 시스템이 자동으로 부여하는 클래스 이름과 엘리먼트 이름이 맵핑된 테이블을 생성한다. 다음의 (그림 7)은 엘리먼트 테이블의 자료 구조를 나타낸다.

```
typedef enum ClassType { NT, EMPTY, ANY };
typedef struct {
        char    *elementName;
        char    *systemName;
        ClassType   type;
        int     nChildren;
        char    **children;
} ClassSymbolTable;
```

(그림 7) 엘리먼트 테이블

둘째, 스키마 생성단계로서 이미 데이터베이스에 있는 SGML 스키마를 임포트하여 이들 클래스로부터 상속 관계를 갖는 DTD스키마를 생성한다. DTD에 존재하는 모든 엘리먼트들에 대해 독립된 클래스를 생성하며, 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. 다음의 <표 3>은 위의 (그림 1) DTD에 대해 스키마 생성기가 생성한 GI 맵핑 테이블이며, (그림 8)은 (그림 1) DTD에 대해 데이터베이스에 자동으로 생성된 C++ ODL(Object Definition Language) 스키마 소스이다.

〈표 3〉 엘리먼트 맵핑 테이블

엘리먼트 이름	시스템이 생성한 클래스 이름
메 모	CL_0
수 신	CL_1
발 신	CL_2
본 문	CL_3
맺음말	CL_4
제 목	CL_5
단 락	CL_6
그 림	CL_7

```
Class CL_0: public RM_NT {
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
```

Element로 부터 상속받은 속성들

NT_Element로 부터 상속받은 속성들

```
    d_List<CL_1> child_1;
    d_List<CL_2> child_2;
    d_List<CL_3> child_3;
    d_List<CL_4> child_4;
)
Class CL_1: public RM_NT {
int nodetype;
d_String name;
d_Ref<RM_Element> parent;
d_Ref<RM_Element> prevSibling;
d_Ref<RM_Element> nextSibling;
d_List<RM_Attribute> attrList;
d_List<RM_EntityRef> entityRefs;
d_List<RM_Element> children;
d_List<RM_PCDATA> PCDATA;
)
Class CL_2 type tuple(
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
    d_List<RM_PCDATA> PCDATA;
)
Class CL_3 type tuple(
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
    d_List<CL_5> child_1;
    d_List<CL_6> child_2;
)
Class CL_4 type tuple(
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
    d_List<RM_PCDATA> PCDATA;
)
Class CL_5 type tuple(
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
    d_List<RM_PCDATA> PCDATA;
```

스키마 생성기에 의해 생성된 속성들

```
)
Class CL_6 type tuple(
    int nodetype;
    name: string
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
    d_List<RM_EntityRef> entityRefs;
    d_List<RM_Element> children;
    d_List<CL_7> child_1;
    d_List<CL_6> child_2;
    d_List<RM_PCDATA> PCDATA;
)
Class CL_7 type tuple(
    int nodetype;
    d_String name;
    d_Ref<RM_Element> parent;
    d_Ref<RM_Element> prevSibling;
    d_Ref<RM_Element> nextSibling;
    d_List<RM_Attribute> attrList;
)
```

(그림 8) 스키마 생성기에 의해 생성된 스키마 소스

마지막 단계는 SGML저장소의 기본 틀을 구축하는 단계로서 스키마 생성 단계에서 마련된 특정 DTD 스키마에 근거하여 베이스를 생성하는 한편, 실제로 SGML Repository 객체 및 DTD관련 정보 등을 저장한다. O2 DBMS에서는 객체를 저장하기 위해 지속성 루트(persistent root)를 먼저 생성하며, 새로운 객체가 생성될 때마다 지속성 루트에 연결시킴으로써 가능하다. 여기서 DTD관련 정보란 DTD에 선언된 엔티티, 표기법, 속성 리스트 등의 정보들을 말하며, 이 정보들은 파서로부터 추출하여 객체를 만든 후 저장한다. 또한 DTD 및 RT파일을 비롯하여 DTD에 정의된 각종 외부 파일도 데이터베이스에 바이너리 형태로 저장한다.

4.2.5 SGML 인스턴스 저장기

SGML 인스턴스 저장기는 저장 관리기로부터 인스턴스를 받아 저장하는 모듈이다. 인스턴스 저장기가 SGML문서를 데이터베이스 클래스로 변환하는 규칙들은 다음과 같다.

변환규칙 1) SGML 데이터의 무결성 보장은 SGML 파서를 구동 시켜 데이터의 무결성을 검증한다.

데이터베이스 스키마의 주요 기능은 유효하지 않은 데이터 입력을 방지하는 것이나, 기존 객체 지향 데이터베이스 시스템들은 SGML의 발생 지시자와 연결자의 의미를 갖는 연산자를 제공하지 못한다[4,5,6,7]. 그러므로 오류가 있는 SGML 데이터의 입력을 데이터베이스의 스키마로 검증할 수 없다. 이의 해결을 위해 본 시스템에서는 데이터를 저장소에 저장하기 전에 SGML 파서를 구동시켜 데이터의 유효성을 검증한다. 기존의 SGML 파서는 SGML 문서의 유효성을 검증한 후 ESIS로 파싱 결과를 출력하나 본 시스템에서는 SGML 문서뿐만 아니라 데이터베이스에 저장된 문서 객체들의 전체 또는 일부분을 검증 할 수 있도록 확장하였다. 즉, 데이터의 유효성은 데이터베이스의 스키마를 이용하지 않고 저장 시스템의 파서에 의한 검증 기능을 통해 검증한 후 저장한다.

변환규칙 2) SGML 문서를 파싱하여 나타나는 트리 상의 모든 노드는 트리 상의 위치관계를 갖는 데이터베이스 클래스로 생성하여 저장한다.

변환규칙 3) 저장소에 저장되는 트리상의 모든 노드는 저장소 객체 식별자(Repository Object Identifier)를 부여하여 트랜잭션의 종료후에도 객체를 영구적으로 식별할 수 있는 기능을 제공한다.

ODMG 객체 모델에서 모든 객체는 데이터베이스내에서 다른 객체와 구분되는 유일한 식별자(Object Identifier)를 가지며, 객체를 참조하는데 사용된다. 그러나 ODMG 객체 모델에서는 객체 식별자의 비트 패턴 구조는 정의하지 않고 있으며, ODBMS 벤더의 구현 문제로 간주한다[10]. 일반적인 ODBMS에서 객체 식별자는 물리적 객체 식별자(physical OID)와 논리적 객체 식별자(logical OID)로 구분된다. 물리적 객체 식별자는 객체의 디스크상의 위치를 나타내며 외부 응용 프로그램에서 접근 할 수 없도록 되어 있다. 논리적 객체 식별자는 외부 응용 프로그램에서 해당 객체의 트랜잭션의 종료 후에도 영구적으로 객체를 참조하는데 사용된다. 그러나 본 시스템의 저장시스템으로 사용하는 O2 ODBMS 버전 4.6에서는 응용 프로그램에서 트랜잭션의 종료 후에도 사용 할 수 있는 논리적 객체 식별자를 제공하지 않으므로 본 시스템에서는 데이터 모델에서 논리적 객체 식별자를 구현하였다[11]. 논리적 객체 식별자는 저장소 식별자, 데이터베이스 식별자, 객체가 O2에 저장될 때 부여되는 SGML 객체의 식별자로 구성된다.

변환규칙 4) "\|", "&", "," 연결자로 지정된 엘

리먼트들의 순서관계는 다중 상속(multiple inheritance)과 다형성(polymorphism)을 이용하여 유지한다.

순서 관계를 유지하기 위해 객체 지향 개념의 다중 상속과 다형성을 이용하여 트리의 노드로 정의될 수 있는 모든 클래스들은 슈퍼클래스로부터 상속 받는 서브클래스를 정의한다. 이 기법을 통해 정의된 서브 클래스들은 ODMG C++ ODL의 컨테이너 클래스인 리스트에 슈퍼 클래스 "RM_Element"를 저장함으로써 순서 관계를 유지 시킨다.

변환규칙 5) SGML 문서에서 나타나는 엔티티 참조는 객체 참조로 변환하여 저장한다.

(그림 2)의 ❷와 ❹에서는 속성값으로 지정된 외부 엔티티와 마크업이 포함된 엔티티가 포함되어 있다. 위의 문서를 의미 손실없이 저장하기 위해 엔티티 참조를 객체 참조로 변환하여 저장한다. 문서의 복원시에는 반대로 객체 참조를 엔티티 참조로 변환하여 엑스포트를 수행한다. 즉, (그림 2)의 ❶에 정의된 Non-SGML 외부 엔티티는 저장소에 객체 식별자를 부여받아 저장되며, ❷의 "그림" 엘리먼트의 속성값으로 참조된 엔티티에서는 저장소에 저장된 객체 식별자를 참조하게 한다.

위의 (그림 2)의 SGML 문서를 인스턴스 저장기가 변환규칙들을 적용하여 저장하는 과정은 다음과 같다.

(1) SGML 문서의 무결성 보장을 위해 변환규칙1을 적용하여 KLESP를 구동시켜 인스턴스 파싱함. 인스턴스를 파싱하여 에러가 없는 경우 RT와 OXF 트리를 생성함.

(2) 트랜잭션을 시작하여 인스턴스 저장을 위한 문서 객체를 생성 한 후, 위의 (그림 2)와 같이 문서 형 선언 부분 집합이 포함된 인스턴스는 엔티티와 표기법 정보, 외부 엔티티의 데이터 파일을 저장함.

(3) 변환규칙2을 적용하여 OXF 트리의 노드를 재귀적으로 순회하며 링크된 RT에서 노드의 타입 정보를 획득한 후 GI 맵핑 테이블에서 대응되는 시스템 클래스 이름을 참조하여 객체를 생성한 후 트리상의 위치정보를 기록함. 예를 들어 (그림 2)의 문서의 루트 엘리먼트 이름이 "메모"이고 노드 타입이 엘리먼트이므로 (그림 9)의 스키마 소스에서 "CL_0" 클래스를 생성함. 생성된 "CL_0"에 변환규칙3을 적용하여 객체 식별자를 부여한 후, 노드 이름, 노드 타입, 등과 속성, 부모노드, 이전 형제노드, 다음 형제노드 등의 정보가 있는 경우 해당 정보를 저장함.

(4) 파스트리 상의 노드 "메모" 엘리먼트의 자식 노드가 있으면 첫번째 자식 노드로 이동하여, 자식 노드의 타입을 보고 (3)의 과정을 반복함. 이때 파스트리의 노드 타입이 서로 다른 자식노드들이 존재하는 비단말 노드인 경우에는 자식노드들의 순서관계를 유지시켜 저장해야 하므로 변환규칙4을 적용하여 데이터베이스 객체의 슈퍼 클래스인 "RM_Element" 클래스를 "children" 리스트에 삽입함. 또한 엔티티 참조인 경우에는 변환규칙5을 적용하여 엔티티 선언 정보를 문서형 선언 부분 집합과 DTD 클래스에서 탐색하여 엔티티 선언 클래스에 대해 1:N 관계를 설정함.

(5) 모든 자식 노드들의 저장이 끝나고 다음 형제 노드가 있는 경우에는 다음 형제 노드로 이동하여 (3)의 과정부터 반복함.

(6) 파스 트리의 모든 노드들을 방문하여 저장이 완료되면 "루트" 엘리먼트 객체를 "문서" 객체에 삽입한 후, 저장 관리기에게 트랜잭션의 성공을 알려주며, 저장 관리기는 스키마 관리기에 저장된 문서에 대한 메타 데이터를 저장하도록 메시지를 전달함.

4.2.6 SGML 객체 관리기

SGML 객체 관리기는 사용자의 요구 또는 질의에 따라 데이터베이스에 프레그먼트로 저장된 SGML 객체들에 대해 다음의 기능들을 수행한다.

- 동적 문서 조합 기능 : 데이터베이스에 저장된 SGML 객체의 문서 전체 또는 일부분을 원래의 SGML 문서로 복원하여 엑스포트
- 색인 포맷 제공 기능 : 색인기가 데이터베이스에 저장된 객체를 색인 할 수 있도록 OID에 의해 명시된 객체의 색인 포맷 제공
- 문서 삭제 기능 : 문서내에 링크된 트리의 모든 노드를 삭제
- DTD 정보 제공 기능 : DTD 내용, RT 정보, 검색을 위한 엘리먼트 이름 제공

- OQL 검색 기능 : 엘리먼트, 속성, 엔티티 등 데이터베이스에 저장된 모든 객체에 대한 OQL 검색

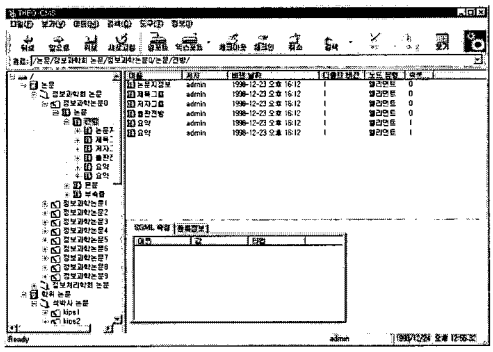
4.3 색인기/검색기 인터페이스

구조 정보 검색은 SGML 문서의 논리적 구조에 대해 검색 범위를 지정한 검색을 말하며, SGML로 전자화된 대규모의 정보에 대해 정확하고 유용하게 검색할 수 있는 효용성을 갖는다. 구조화된 SGML 정보의 검색은 기존의 내용 중심의 정보 검색과는 달리 다음과 같은 추가적인 기능들이 요구된다. 첫째, SGML 문서의 정보 표현의 기본 단위는 엘리먼트이다. 따라서, 기존의 정보 검색에서의 문서 단위 검색과는 달리 엘리먼트 단위의 검색이 이루어져야 한다. 둘째, 구조화된 문서는 태그와 SGML 데이터가 DTD에 정의된 논리적 구조에 의해 혼합되어 있다. 그러므로 구조화된 문서를 효율적으로 검색하기 위해서는 문서의 논리적 구조 내에서의 특정 SGML 엘리먼트와 데이터의 위치, 속성값 등을 표현할 수 있는 인덱스 구조가 정의되어야 하며, 키워드 인덱스이외에 구조 정보에 대한 인덱스가 필요하다. 셋째, 기존의 내용만을 고려한 정보 검색의 질의 언어로는 문서의 구조에 대한 질의 표현을 나타낼 수 없으므로, 문서의 내용 및 구조에 대한 질의 표현을 나타낼 수 있는 질의어의 정의가 필요하다 [12,13,14].

본 실험실에서는 데이터베이스에 저장된 객체를 색인하여 O2의 하부 저장 시스템인 O2Store에 색인 정보를 저장하는 색인기와 검색 기능을 제공하는 검색기를 구현하였다[14]. 저장 시스템에서 색인과 검색을 지원하기 위해 다음의 기능을 제공한다. 색인을 지원하기 위해 데이터베이스에 저장된 문서중에서 색인되지 않은 문서의 목록을 보관하여 색인기의 색인 요청이 발생하는 경우, 데이터베이스 객체를 색인기에 전송한다. 색인기는 구조 정보, 키워드, 객체 식별자 등을 색인 테이블에 저장한다. 또한 검색기에서 문서나 엘리먼트 부트리를 요구하는 경우에는 OID로 지칭된 객체를 OQL 질의를 수행하여 검색한 후 해당 정보를 반환한다. 구조 검색 시스템에 대한 자세한 내용은 참고문헌 [13,14]에서 기술하고 있다.

4.4 구현 환경

본 논문에서 구현한 저장시스템의 플랫폼은 SUN UltraSPARC2에서 O2 ODBMS 4.6버전을 사용하였다. 사용한 운영체제와 컴파일러는 Solaris 2.5.1과 SUN C++ 4.1를 사용하였다. 개발한 시스템은 클라이언트 응용 프로그램 개발을 위한 C++ SDK를 제공하며, 아래의 (그림 9)는 SDK를 이용하여 개발된 클라이언트 워크벤치인 RM 네비게이터에서 저장소에 저장된 문서 인스턴스를 브라우징한 화면이다.



(그림 9) 저장된 문서의 브라우징 화면

5. 결 론

본 논문에서는 SGML 문서 저장을 위한 데이터 모델 및 저장 시스템의 구현에 대해 기술하였다. 제안한 데이터 모델은 ODMG 객체 모델에 기반하므로 ODMG 표준을 수용하는 O2이외의 다른 ODBMS에도 쉽게 이식될 수 있는 장점이 있다. 또한 인스턴스의 트리상에서의 순서 관계를 유지 시킬 수 있으며, 데이터의 삽입, 삭제, 변경시 발생 지시자와 연결자의 의미에 따른 유효성 검증은 SGML 파서를 이용하여 해결하였다. 이외에도 논리적 객체 식별자를 구현하여 문서내에 발생된 외부 엔티티 참조를 객체 참조로 변환하여 의미 손실을 방지한다. 구현한 저장 시스템은 제안한 데이터 모델에 따라 DTD의 구조를 반영하는 스키마를 동적으로 생성하며, 문서 인스턴스를 정보의 손실 없이 저장, 검색 할 수 있다. 또한 데이터베이스에 저장된 문서 정보들을 색인 및 검색 할 수 있는 API들을 제공한다. 서버와 연동하여 사용할 수 있는 인터페이스를 제공하므로 대용량의 전자 문서를 저장하고 검색하는 다양한 응용 분야에 사용될 수 있다.

향후 연구 방향은 다수 사용자의 공동 저작, check-in/check-out, 버전 제어, 워크플로우 등을 지원하는 통합 문서 관리 시스템의 기능들을 구현하는 것이다.

또한 DSSSL를 이용한 포맷팅 엔진의 통합 및 웹 서비스를 위한 웹 서버의 구현, 그리고 XML의 링크 지원 언어인 XLink(XLink and XPointer Languages)을 지원하는 기능을 연구 중에 있다.

참 고 문 헌

[1] C. F. Goldfarb and Yuri Rubinsky, The SGML Handbook, Clarendon Press, Oxford, 1990.

[2] W3C XML Working Group, Extensible Markup Lanugage (XML), Available via http://www.w3c.org/XML, 1998.

[3] Arbortext, Inc. Getting Started with SGML, Available via http://www.arbortext.com/wp.html, 1995.

[4] V. Christophides, et al, "From Structured Documents to Novel Query Facilities," ACM SIGMOD, pp.313-324, Minesota, USA, 1994.

[5] G. E. Blake, M. P. Consens, P. Kilpelainen, P. A. Larson., T. Sinder, and F. W. Tompa, "Text/Relational Database Management Systems : Harmonizing SQL and SGML," In Proceedings of the International Conference on Applications of Databases, Vadstena, Sweden, 1994.

[6] R. Sacks-Davis, T. Arnold-Moore, and J. Zobel, "Database Systems for Structured Documents," In International Symposium on Advanced Database Technologies and Their Integration, pp. 272-283, Nara, Japan, 1994.

[7] Patricia Francois, "Generalized SGML Repositories : Requirements and modelling," Computer Standards and Interfaces 18, 1996.

[8] Texcel N.V., "Information Manager Programmer's Guide", Texcel, 1997.

[9] 한에노, 박인호, 강현석, 김완석, "SGML 문서의 관리를 위한 객체지향 데이터베이스 설계", 한국정보처리학회 논문지 제4권 제3호, pp.670-684, 1997.

[10] R.G.G. Cattell, The Object Database Standard : ODMG-93, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[11] O2 Technology, ODMG C++ Binding Guid, Release 4.6, December 1996.

[12] 김현기, 이상기, 주종철, 박동인, "객체 지향 SGML 저장시스템의 설계 및 구현", 한국정보처리학회 추계 학술 발표 논문집 제4권 2호, pp.387-390, 1997.

[13] 이수현, 김문석, 김은식, 주종철, "구조 정보 검색을 위한 시험용 질의 모음 개발", 한글 및 한국어 정보처리 학술대회", 1997.

[14] 손정한, 이희주, 장재우, 심부성, 주종철, "구조화 문서를 위한 정보검색 인덱스의 구현과 성능 평가", 한국정보과학회 봄 학술발표논문집 제25권 1호, pp.473-475, 1998.

김 현 기

e-mail : hkk@etri.re.kr

1994년 전북대학교 컴퓨터공학과 (학사)

1996년 전북대학교 컴퓨터공학과 (공학석사)

1996년~현재 한국전자통신연구원 연구원

관심분야 : SGML/XML 문서 관리 시스템, XML EDI, STEP-SGML

노 대 식

e-mail : rohsik@etri.re.kr

1994년 경북대학교 전자계산학과 (학사)

1996년 경북대학교 컴퓨터과학과 (이학석사)

1996년~현재 한국전자통신연구원 연구원

관심분야 : SGML/XML, DSSSL, XML EDI

강 현 규

e-mail : hkkang@etri.re.kr

1985년 홍익대학교 전자계산학과 (학사)

1987년 한국과학기술원 전산학과 (공학석사)

1992년 정보처리 기술사 자격 취득

1997년 한국과학기술원 전산학과 (공학박사)

1987년~현재 한국전자통신연구원 선임연구원, 문서정보연구팀장

관심분야 : 정보 검색, 자연언어 처리, SGML/XML, 지식 정보