WWW 데이터베이스 인터페이스를 위한 UCM(United CGI Management) 시스템의 설계

김 은 경† · 황 병 연††

요 약

최근 들어 WWW(World Wide Web)와 데이터베이스의 통합 문제가 큰 관심이 되고 있다. 본 논문에서는 WWW와 데이터베이스 시스템을 연동하기 위하여 기존의 서버나 클라이언트를 변경하지 않고 그대로 이용할 수 있는 새로운 방법으로서 CGI(Common Gateway Interface) 방식에 기반을 둔 UCM(United CGI Management) 시스템을 제안한다. CGI 방식 중에서 CGI 실행 방식은 데이터베이스에 접근할 때마다 CGI 프로그램을 실행시키기 때문에 데이터베이스에 대한 많은 동시 요구가 발생될 경우에는 시스템 성능이 크게 저하된다. 하지만 UCM 시스템은 데몬 방식인 통합된 CGI를 사용하여 이러한 CGI 실행 방식의 문제점을 개선한다. 또한 제안된 시스템은 DBMS(DataBase Management Systems)를 통하지 않기 때문에 SQL 언어를 사용하지 않으며 WWW 데이터베이스를 더 경제적인 비용으로 이용할 수 있다. 본 논문에서는 WWW 데이터베이스 인터페이스를 구축하기 위해 확장된 HTML 태그를 정의하고 데이터베이스로의 접근을 위해 EXODUS 저장 관리자(storage manager)를 사용한다.

Design of the UCM(United CGI Management) System for WWW Database Interface

Eun-Kyoung Kim† · Byung-Yeon Hwang††

ABSTRACT

Many people are the latest interested in the integration of the WWW and database systems. The UCM(United CGI Management) system, which is proposed on this study, is based on the method using CGI that does not change existing servers and clients to integrate the WWW and database system. Since the CGI executable architecture executes a CGI program whenever the database is accessed, the system performance is reduced greatly when many requests to the database are made simultaneously. However, the UCM system improves the system performance since it uses the integrated CGI running a daemon process. Moreover, the proposed system can use the WWW database more economically and doesn't use the SQL because it does not use a DBMS. In UCM system, we define the extended HTML tags to implement the WWW database interface and use the EXODUS storage manager for accessing the database.

1. 서 론

최근 들어 인터넷 사용자가 WWW 데이터베이스를 이용하는 경우와 동적인 문서의 작성에 사용될 데이터를 데이터베이스로 보관하고자 하는 요구가 증가함에 따라 WWW와 데이터베이스의 통합 문제가 큰 관심의 대상이 되고 있다.

WWW 상에서 데이터베이스로 접근하기 위한 방법

† 정 회 원 : 가톨릭대학교 컴퓨터공학과
†† 종신회원 : 가톨릭대학교 컴퓨터공학과 교수
논문접수 : 1998년 2월 23일, 심사완료 : 1999년 6월 21일

은 데이터베이스를 접속하는 프로그램이 위치하는 장소와 CGI(Common Gateway Interface) 프로그램의 사용 여부에 따라서 CGI 방식, 서버 확장 방식, 클라이언트 확장 방식으로 분류된다[1]. 여기서 CGI란 기존의 웹 서버에 의해 지원되는 것으로 웹 서버와 서버 호스트 컴퓨터의 자원을 연결하는 인터페이스를 의미한다. 이러한 CGI 기능을 이용하는 CGI 방식은, 각각 기존의 서버와 클라이언트를 변경해야 하는 서버 확장 방식과 클라이언트 확장 방식에 비하여, 구조가 간단하고 기존의 웹 서버, 웹 브라우저, URL, HTTP, HTML 문서 등을 그대로 사용할 수 있어서 범용적이라는 큰 장점이 있다. 하지만 CGI 방식의 한 종류인 CGI 실행 방식에서는 데이터베이스 응용 프로그램이 CGI 실행 파일로 작성되어 데이터베이스 요구가 있을 때마다 생성, 종료되기 때문에 많은 요구가 동시에 발생될 경우에는 시스템 성능이 크게 저하된다. 그리고 이러한 문제점을 해결하기 위해 대부분 DBMS를 이용하기 때문에 일반 인터넷 사용자 입장에서는 경제적인 부담과 관리 능력의 부재로 인해서 어려운 점이 많다.

따라서 본 논문에서는 DBMS를 사용하지 않으면서 CGI 방식에 기반을 둔, 인터넷을 위한 WWW 데이터베이스 인터페이스를 설계하는 새로운 방법을 제안한다. 본 연구의 핵심적인 특징은 다음과 같다.

- 데몬(daemon) 방식으로 운영되는 통합된 CGI
- 데이터베이스로의 접근 기능을 가지는 확장된 HTML 태그
- EXODUS 저장 관리자(storage manager)[2, 3]

본 논문의 구성을 보면, 2장에서는 WWW와 데이터베이스를 연동하는 여러 방식과 관련 사례를 소개 및 비교하고, 3장에서는 본 논문에서 제안한 CGI 프로그램을 이용하여 WWW와 데이터베이스를 연동하는 개선된 시스템의 특징, 구조 및 사용자 인터페이스를 설명한다. 4장에서는 개선된 시스템 구축을 위한 프로그래밍 설계에 관한 내용을 기술하며, 마지막으로 5장에서는 결론과 향후 연구 방향을 제시한다.

2. WWW와 데이터베이스의 연동 구조

WWW와 데이터베이스와의 연동 구조는 데이터베이스 접속 프로그램이 서버쪽에 있지만 CGI 프로그램을 사용하여 데이터베이스를 연결함으로써 기존의 서버를 그대로 이용할 수 있는 CGI 방식, 데이터베이스 접속 프로그램이 서버쪽에 위치하면서 CGI 프로그램을 사용하지 않고 서버를 확장하는 서버 확장 방식, 데이터베이스 접속 프로그램이 클라이언트쪽에 위치하면서 클라이언트를 확장하는 클라이언트 확장 방식 이렇게 크게 세 가지로 분류할 수 있다.

첫 번째로 CGI 방식은 CGI 실행 방식과 CGI 응용 서버 방식으로 분류된다. CGI 실행 방식은 CGI 실행 파일을 사용하여 데이터베이스로 접속하는 것으로 ORDBMS(Object Relational DBMS)인 일러스트라(Illustra)의 Web DataBlade[4] 모듈(module)을 예로 들 수 있다. 그리고 CGI 응용 서버 방식은 디스패처(dispatcher)와 데몬 방식의 데이터베이스 응용 프로그램을 사용하는 방식인데, 그 예로는 WWW와 ORDBMS인 UniSQL/X와의 연동을 제공하는 UniWeb[5], RDBMS(Relational DBMS)인 오라클(Oracle) 데이터베이스 시스템과의 연동을 제공하는 오라클 웹 서버(Oracle WebServer)[6, 7]가 있다. 최근에는 앞에서 기술한 제품들이 계속 버전 업 되면서 다양한 기능과 방식의 추가로 자체내의 단점들을 보완하고 있는 추세이다. 예를 들어, 최근의 일러스트라에서는 데몬을 이용하며 확장 API(Application Programming Interface)를 지원한다. 또한 오라클 웹 서버와 UniWeb의 최근 버전에서도 확장 API를 지원한다.

두 번째로 서버 확장 방식은 확장 API를 이용하여 데이터베이스에 접근하는 확장 API 방식과 웹 서버 내에 특정 DBMS를 직접 접속할 수 있는 데이터베이스 게이트웨이 기능을 구현한 전용 서버 방식으로 분류된다. 서버 확장 API의 예로는 넷스케이프의 NSAPI(Netscape Server API)[8], 마이크로소프트의 ISAPI(Internet Server API)[9]를 들 수 있다.

마지막으로 클라이언트 확장 방식은 클라이언트쪽에 있는 외부 뷰어(external viewer)를 이용하여 데이터베이스에 접속하는 외부 뷰어 방식과 외부 뷰어의 기능을 웹 브라우저에 포함시킴으로써 브라우저 자체에서 데이터베이스로의 접근 기능을 지원하는 브라우저 확장 방식으로 나눌 수 있다.

한편 본 논문에서 기반으로 하는 CGI 방식의 세 가지 사례들에 대해 살펴보면 다음과 같다.

■ Illustra Web DataBlade

일러스트라 서버로 삽입되는 객체 확장 모듈인 여러

가지 DataBlade들 중에서 데이터베이스와 웹 서버의 연동을 위한 동적 웹 인터페이스 프로그램인 Web DataBlade는 요구에 해당하는 데이터베이스 내의 응용 페이지를 분석하여 처리한 후에 그 결과인 HTML 문서가 클라이언트로 전해지게 한다. 여기서 응용 페이지란 데이터베이스 내에 저장된 것으로 SQL, 조건부 논리 등을 포함하는 HTML 문서를 의미하며, 일러스트라 Web DataBlade의 수행을 위하여 웹 서버에 의해서 실행되는 CGI 드라이버인 일러스트라 웹 드라이버와 웹 데몬이 사용된다.

■ **UniWeb**

UniWeb에서 디스패처는 필요한 UniTCL(Tool Command Language) 서버의 상태 정보를 bulletin board에서 얻음으로써 해당 UniTCL 서버로 요구를 전달한다. 이때 디스패처는 CGI용 실행 파일이기 때문에 데이터베이스에 접근할 때마다 새로이 실행되지만 프로세스의 크기가 매우 작아서 시스템 성능 저하 문제를 발생시키지 않는다. 그리고 UniTCL 서버는 데몬으로 동작하기 때문에 한 번 UniSQL/X 데이터베이스에 접속하면 계속 그 접속을 유지한다.

■ **Oracle WebServer**

오라클 웹 서버에서의 WWW와 데이터베이스의 연동 과정은 다음과 같다. 먼저 CGI 기술을 사용하며 데몬 방식으로 운영되는 오라클 웹 리스너(Web Listener)가 디스패처를 통하여 WRB(Web Request Broker)로 요구를 넘겨주고, 비동기(asynchronous) 요청 취급자(handler)인 WRB는 전달받은 요구가 적절한 카트리지(cartridge)를 통하여 데이터베이스로 전달되게 한다. 여기서 확장 가능한 객체인 각각의 카트리지는 CORBA(Common Object Request Broker Architecture)[10] 기반의 객체 버스인 ICX(Inter-Cartridge eXchange)를 통하여 다른 카트리지나 클라이언트, 서버 등에 접근할 수 있다. 이와 같이 요구에 대한 한 가지 서비스를 제공한 오라클 웹 서버는 다음 요청을 위하여 데이터베이스와의 연결을 계속 유지하게 된다.

<표 1>에서 살펴본 바와 같이, 서버와 클라이언트 확장 방식의 경우는 각각 기존의 서버나 클라이언트를 변경해야 한다는 부담과 표준화의 문제로 인하여 범용적으로 사용되기가 어렵다. 그리고 앞에서 기술한 모든 경우는 고가의 DBMS를 사용해야 한다는 점에서 본 논문에서 제안하는 시스템과 큰 차이가 있다. 이러한 차이점을 본 논문에서 CGI 방식의 대표적인 세 가지 예로 든 제품들과 비교하여 살펴보면 다음과 같다.

〈표 1〉 데이터베이스 게이트웨이의 비교

		장 점	단 점
CGI 방식	CGI 실행 방식	• 기존의 웹 서버 및 웹 브라우저, URL, HTTP, HTML 문서를 변경하지 않고 사용하는 것이 가능함 • 확장이 용이함 • 다양한 프로그래밍 언어의 사용이 가능함	• 클라이언트로부터 동시 요청이 많으면 시스템의 성능 저하를 초래함
	CGI 응용 서버 방식	• CGI 실행 방식의 장점을 그대로 가짐 • CGI 실행 방식에서의 시스템 성능 저하 문제를 해결함	• 디스패처와 데몬 프로그램의 구현이 요청되어 CGI 실행 방식의 구조보다 복잡함
서버 확장 방식	확장 API 방식	• CGI 실행 방식에서의 시스템 성능 저하 문제를 해결함 • 기존의 웹 브라우저, URL, HTTP, HTML 문서를 그대로 사용 가능함	• 기존의 웹 서버를 변경해야 함 • 별도의 확장 API를 사용한 루틴 작성의 어려움으로 인한 확장성 문제 • 웹 서버 확장 API에 대한 표준의 부재로 다른 웹 서버와의 호환성이 좋지 않음
	전용 서버 방식	• 확장 API 방식의 장점을 그대로 가짐 • 서버가 직접 데이터베이스를 다룸으로 인하여 효율성이 좋음	• 기존의 웹 서버를 변경해야 함 • 특정 웹 서버를 사용해야 하므로 다른 웹 서버와의 호환성 및 확장성이 좋지 않음
클라이언트 확장 방식	외부 뷰어 방식	• 다양하고도 편리한 사용자 쉽의 화면 구성 • 여러 가지 개발 도구를 사용함으로써 개발 효율을 높일 수 있음	• 기존의 웹 브라우저를 변경해야 함 • 외부 뷰어와 클라이언트용 데이터베이스 접속 프로그램에 대한 표준이 없음 • 외부 뷰어를 이용하므로 사용자가 웹 브라우저 이외의 별도의 소프트웨어 기능을 익혀야 함
	브라우저 확장 방식	• 사용자 쉽의 화면의 다양성 • 외부 뷰어의 기능을 웹 브라우저에 포함시켜 하나의 통합된 환경에서 사용자에게 서비스를 할 수 있음	• 기존의 웹 브라우저를 변경해야 함 • 클라이언트쪽의 확장 부분인 스크립트 번역기와 데이터베이스 게이트웨이에 대한 표준이 없음

먼저 UniWeb은 데이터베이스 응용 프로그램을 구성하는 UniTCL 명령어에, SQL을 확장한 UniSQL/X 시스템의 데이터 언어인 SQL/X 문의 수행 명령어를 포함함으로써 WWW와 UniSQL/X를 연동한다. 오라클 웹 서버는 여러 가지 카트리지 중에서 가장 핵심적인 PL/SQL 카트리지를 통하여, 데이터베이스 내에 있으면서 SQL 문을 포함할 수 있는 PL/SQL 저장 프로시저를 실행함으로써 HTML 문서를 생성한다. 한편 일러스트라 Web DataBlade의 경우는 태그를 추가하여 활용함으로써 HTML 문서 내에 SQL 문장을 삽입하여 DBMS를 사용하게 된다.

이에 비하여 본 논문에서 제안하는 시스템은 CGI 방식에 기반을 두지만 DBMS를 통하지 않기 때문에 SQL 문장을 사용하지 않는다. 그 방법으로 본 논문의 시스템에서는 데이터베이스 접근에 관련하여 새로운 태그를 정의하는데, 이렇게 확장된 태그는 일러스트라의 Web DataBlade 모듈에서 지원하는 태그와는 차이가 있다. 구체적으로 보면, 일러스트라 Web DataBlade 모듈에서는 SQL 문장을 포함하는 별도의 태그인 MISQL을 지원하는 반면에 본 논문에서는 검색, 삽입, 삭제, 수정 등의 기능을 수행하는 각각 다른 태그를 정의하여 기존의 SQL 문이 담당하던 데이터베이스의 데이터 조작의 역할을 대신한다.

한편 이러한 본 논문에서의 확장된 HTML 태그는 최근에 W3C(World Wide Web Consortium)에서 발표한 XML(eXtensible Markup Language)과는 큰 차이점이 있다. XML은 문서 포맷을 정의하는 HTML의 단순함의 문제를 보완하는 역할을 하지만 데이터베이스로 접근할 수 있는 기능을 포함하지 않는다.

3. UCM 시스템의 설계

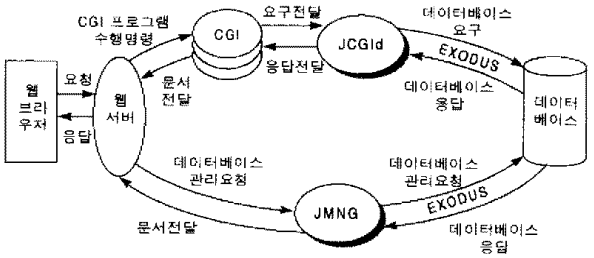
3.1 UCM 시스템의 특징

실제적으로 기존의 상용화된 DBMS는 보안 유지에 대한 부분 및 DBMS 자체적인 데이터베이스의 관리 능력면에 있어서 뛰어난 여러 가지 기능을 가지고 있지만 사용자 인터페이스 부분 등에서 그 규모가 크기 때문에 상당히 고가이다. 즉 이러한 점을 고려하면 앞의 2장에서 CGI 방식의 예로 든 세 가지에서는 모두 SQL 문장을 사용함으로써 DBMS를 이용하게 되어 사용자에게 부담을 주게 된다. 따라서 DBMS를 사용하지 않고 사용자가 데이터베이스로 접근 할 수 있도록 하기 위하여 본 논문에서 제안한 시스템에서는 웹 인터페이스를 제공하는 경제적인 도구(tool)를 사용한다. 본 논문에서는 이러한 시스템을 UCM(United CGI Management) 시스템이라고 명명한다. 다음은 UCM 시스템의 특징이다.

- CGI 방식에 기반을 두므로 기존의 웹 서버, 웹 브라우저 등을 그대로 사용하는 것이 가능하다.
- 데이터베이스로의 요구가 있을 때마다 생성, 종료되는 것이 아니라 계속 요구를 기다리면서 데몬 방식으로 운영되는 통합 CGI 프로그램인 JCGId를 사용함으로써 CGI 실행방식에서의 시스템 성능 저하 문제를 개선할 수 있다.
- 기존의 CGI 방식을 사용하는 시스템들과 달리, SQL 언어를 이용하지 않고 데이터베이스 접근 기능을 가지는 확장된 HTML 태그를 새로이 정의하여 이용한다. 즉 이러한 확장된 태그는, 일반 인터넷 사용자들이 이용하기에는 관리 및 경제면에서 부담이 되는 DBMS를 통하지 않고 데이터베이스의 데이터 조작에 대한 클라이언트의 요구를 매개변수를 통하여 JCGId로 넘겨줌으로써 프로그램이 실행되게 한다.
- CGI와 데이터베이스를 연결하는 인터페이스로는 EXODUS 저장 관리자를 이용한다.

3.2 UCM 시스템의 구조

본 논문에서 제안된 시스템은 크게 데이터베이스의 데이터를 조작하는 JCGId 프로그램 부분과 데이터베이스의 테이블을 관리하는 JMNG 프로그램 부분으로 나누어진다. UCM 시스템에서의 WWW와 데이터베이스의 연동 구조는 (그림 1)과 같다.



(그림 1) UCM 시스템

먼저 데이터베이스의 데이터를 조작하는 부분을 보면 다음과 같다. 즉 본 논문에서 확장한 HTML 태그가 이용된 HTML 문서의 형식으로 클라이언트가 웹 서버에게 데이터베이스에 대한 요구를 하면 웹 서버가

CGI 프로그램 수행 명령을 내린다. 그리고 CGI 프로그램이 전달받은 요구를 해석하여 데이터베이스 데이터에 대한 접근 종류를 판단하고 요구를 통합된 CGI 데몬 프로그램인 JCGId로 넘긴다. 그러면 JCGId가 EXODUS를 호출함으로써 데이터베이스로 접근하여 각각의 요구들을 수행한 결과를 JCGId를 거쳐서 CGI로 전달한다. 마지막으로 클라이언트가 CGI 프로그램과 웹 서버를 통하여 데이터베이스의 응답 결과를 받는다.

한편 데이터베이스의 테이블을 관리하는 부분을 보면, 먼저 클라이언트가 편리한 HTML FORM 화면으로 데이터베이스 관리에 대한 요구를 한다. 그러면 JMNG는 웹 서버를 통하여 요구를 받아서 그 종류에 따라 EXODUS를 이용함으로써 데이터베이스 응답 결과를 HTML 문서의 형태로 클라이언트쪽으로 보내 준다.

다음은 UCM 시스템을 구성하는 부분 중에서 가장 핵심적인 JCGId와 JMNG 프로그램의 역할 및 각 프로그램이 클라이언트와 데이터를 상호 교환하는 것에 대한 내용을 정리한 것이다.

- JCGId : 데이터베이스의 데이터 조작
  입력 - 테이블명, 조건, 컬럼명의 리스트, 컬럼의 순서, 컬럼의 값
  출력 - HTML 문서
- JMNG : 데이터베이스의 테이블 관리
  입력 - 테이블명, 컬럼수, 컬럼명, 데이터 타입, 기본키
  출력 - HTML 문서

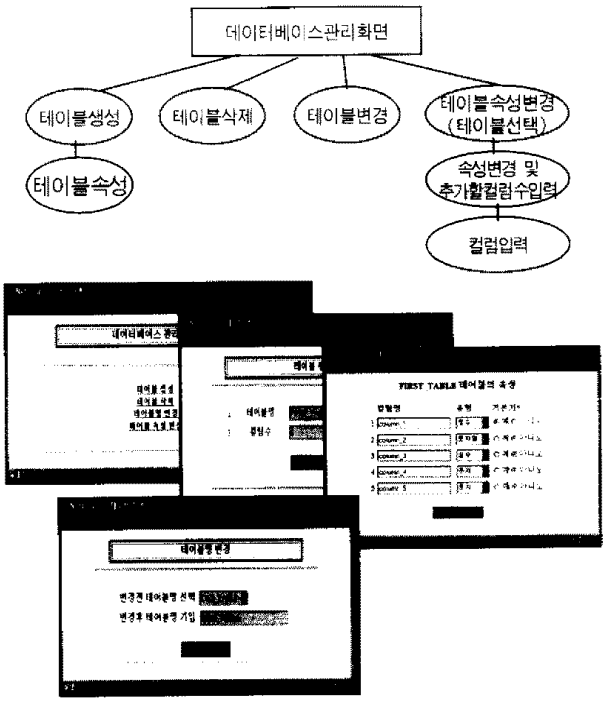
3.3 확장된 HTML 태그

본 절에서는 클라이언트가 데이터베이스의 데이터를 편리하게 조작할 수 있게 하는 확장된 HTML 태그를 정의한다. 각 태그들의 시작 시에 나타나는 +는 구분자로서, + 다음에 나타나는 태그가 데이터베이스로의 접근에 관련하여 본 논문에서 확장된 태그임을 CGI 프로그램이 알게 한다.

- **변수 할당** <+JVAR 속성들> 값 또는 변수들 <+/JVAR>
  속 성 : NAME=변수명, DEFAULT=default값, FUNCTION =함수명
  함수명 : URLENCODE( ), URLDECODE( ), REPLACE( ), SUBSTRING( )
- **검 색** <+JSEL 속성들>
  속 성 : TAB=테이블명, LIMIT=(조건), 컬럼명들, SORT=(컬럼명 [DESC])들
- **삽 입** <+JINS 속성들>
  속 성 : TAB=테이블명, (컬럼명=값)들
- **삭 제** <+JDEL 속성들>
  속 성 : TAB=테이블명, LIMIT=(조건)
- **수 정** <+JUPD 속성들>
  속 성 : TAB=테이블명, LIMIT=(조건), SET (컬럼명=값)들
- **조건에 따른 처리1** <+JIF 속성> 처리문 <+/JIF>
  속 성 : LIMIT=(조건)
- **조건에 따른 처리2** <+JWHILE 속성> 처리문 <+/JWHILE>
  속 성 : LIMIT=(조건)
- **에러 처리** <+JERR ERRNO=에러코드번호> 처리문 <+/JERR>

3.4 데이터베이스 설계 사용자 인터페이스

(그림 2)는 UCM 시스템에서의 데이터베이스 설계 사용자 인터페이스의 전체적 구조 및 각 화면 중 일부를 나타낸 것이다. 사용자는 데이터베이스 관리 화면을 통하여 데이터베이스를 설계하기 위한 다양한 항목을 선택할 수 있다.

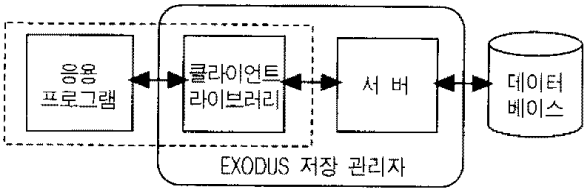


(그림 2) 데이터베이스 설계 사용자 인터페이스

4. UCM 시스템 구축을 위한 프로그램 설계

4.1 EXODUS의 개요 및 구조

1980년대 이후에 Wisconsin 대학에서 개발된 EXODUS는 EXODUS 저장 관리자와 E 프로그래밍 언어를 위한 컴파일러, 그리고 이들 각각에 대한 문서와 테스트 프로그램을 포함하는 소프트웨어이다. 본 논문에서는 EXODUS 저장 관리자를 이용하여 UCM 시스템을 설계한다. 데이터베이스를 관리하는 EXODUS 저장 관리자는 바이트들의 묶음으로 이루어진 다양한 길이의 저장 객체(storage object)를 기본 단위로 하는 다중 사용자 객체 저장 시스템(multi-user object storage system)인데 다음의 (그림 3)과 같이 클라이언트-서버 구조로 되어있다.



(그림 3) EXODUS 저장 관리자

저장 관리자의 '클라이언트' 또는 '클라이언트 라이브러리'라고 불리는 부분은 응용 프로그램과 연결된 함수들의 라이브러리이다. 여기서 응용 프로그램들이 클라이언트 라이브러리 안에 있는 함수들을 호출함으로써 클라이언트내의 버퍼 풀에 놓인 객체상에서 어떤 연산이 수행된다. 한편 저장 관리자의 서버 부분은 클라이언트 부분과 협력하는 UNIX 프로세스들의 집합이다. 이러한 서버와 클라이언트 사이의 인터페이스는 디스크 블럭들을 할당하고 회수하는 것과 같은 저수준의 서비스와 트랜잭션 관리를 수행하는 원격 프로시져들의 집합으로 이루어진다.

4.2 EXODUS 저장 관리자 응용 인터페이스

다음은 EXODUS 저장 관리자 응용 인터페이스에서 가장 핵심적인 함수들이다.

- **초기화와 종료 연산**

  sm_Initialize( ), sm_ShutDown( )

- **트랜잭션 연산**

  sm_BeginTransaction(tid), sm_CommitTransaction(tid), /* tid : transaction identifier */

- **버퍼 연산**

  sm_OpenBufferGroup(groupSize, policy, groupIndex, flags), sm_CloseBufferGroup(groupIndex)

- **객체들상에서의 연산**

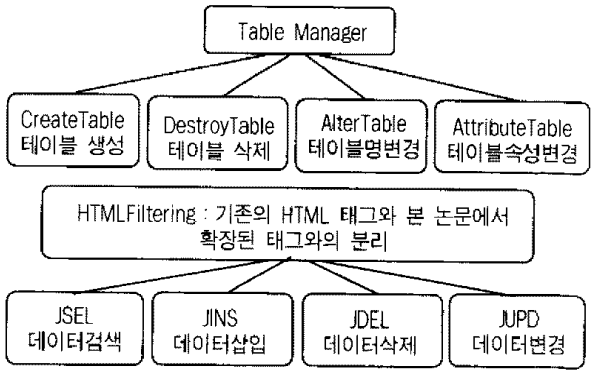
  sm_CreateObject(groupIndex, fid, nearHint nearObj, objHdr, length, data, oid),
  sm_DestroyObject(groupIndex, oid),
  sm_WriteObject(groupIndex, start, length, data, userDesc, release), /* fid : file identifier, oid : object identifier, objHdr : object header, userDesc : user descriptor */

- **파일상에서의 연산**

  sm_CreateFile(groupIndex, volid, fid), sm_DestroyFile (groupIndex, fid),
  sm_OpenScan(fid, type, groupSize, scanDesc, oid), sm_ScanNextObject(scanDesc, start, length, retDesc, eof), sm_CloseScan(scanDesc)

4.3 UCM 프로시져

본 논문에서 제안된 UCM 프로시져의 각 기능과 구조는 다음과 같다.



(그림 4) UCM 프로시져

■ 테이블 생성 프로시져

CreateTable 함수는 테이블명과 컬럼수 그리고 기본키 컬럼을 입력받아서 전달받은 수 만큼의 컬럼들이 존재하는 테이블을 생성하고 기본키(primary key) 컬럼을 표시한다. 또한 컬럼명과 타입의 쌍들로 구성된 columns를 받아서 해당 테이블에 입력한다.

CreateTable(tname, colno, columns, key)
/* tname : IN table name

```
    colno          : IN column number
    columns        : IN set of (column, type)
    key            : IN primary key */
}
sm_Initialize( ); /* 시상 관리자의 데이터 구조들을 초기화한다. */
sm_BeginTransaction(tid); /* *tid : OUT transaction identifier,
    트랜잭션을 시작한다. */
sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);
  /* groupSize     : IN the maximum group size in pages
    policy         : IN the group's replacement policy
    *groupIndex    : OUT the group's index
    flags          : IN buffer group attributes
    새로운 buffer group을 open한다. */

sm_CreateFile(groupIndex, volid, fid);
  /* groupIndex    : IN buffer group in use
    volid          : IN the volume in which to place the file
    *fid           : OUT the file identifier of the new file
    volid 인수에 의해 확인된 볼륨상에 새로운 파일을 생성한다. */
CreateFile(TABLES); /* CreateFile(TABLES)는 UNIX 파일 시
    스템 환경을 이용하여 TABLES 파일을 생성한다. TABLES
    파일은 sm_CreateFile로부터 전달받은 fid와 입력인 tname,
    colno, 그리고 key 컬럼을 이용하여 구성되는데, TABLES 파
    일이 이미 생성되어 있는 경우에는 각 구성 원소만 파일에 추
    가시킨다. 다음은 TABLES 파일의 구조이다.
    TABLES
```

fid	Table Name	Column No.	Key Column
1	A	2	1
2	B	3	2
:	:	:	:

```
    */
CreateFile(COLUMNS); /* CreateFile(COLUMNS)는 UNIX 파일
    시스템 환경을 이용하여 COLUMNS 파일을 생성한다.
    COLUMNS 파일은 입력인 columns, 즉 (column, type) 쌍과
    TABLES 파일로부터 얻은 fid로 구성된다. 이때 COLUMNS
    파일이 기존하는 경우에는 각 구성 원소만 파일에 추가시킨다.
    다음은 COLUMNS 파일의 구조이다.
    COLUMNS
```

fid	Column Name	Type
1	Acol_1	int
1	Acol_2	char
2	Bcol_1	string
2	Bcol_2	int
2	Bcol_3	char
:	:	:

```
    */
sm_CloseBufferGroup(groupIndex); /* groupIndex : IN the group
    being closed.
    groupIndex 인수에 의해 확인된 open buffer group을 close
    한다. */
sm_CommitTransaction(tid); /* tid : IN transaction identifier.
    tid 인수에 의해 확인된 트랜잭션의 효과를 commit한다. */
sm_ShutDown( ); /* 모든 open buffer group들의 close한다. 그리
    고 클라이언트 라이브러리에 의    해 실행 시간에 할당된 메
    모리를 자유롭게 한다. */
}
```

■ 테이블 삭제 프로시저

DestroyTable은 입력받은 테이블명에 해당하는 테이블을 삭제하는 함수이다.

```
DestroyTable(tname)
    /* tname : IN table name */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);
        fid = Search_TableFile(tname); /* TABLES 파일에서, 입력
            으로 들어온 테이블명인 tname에 해당하는 fid를 찾아
            서 반환한다. */
        sm_DestroyFile(groupIndex, fid); /* 전달받은 fid에 해당하는
            테이블을 삭제한다. */
        DestroyFile(fid); /* TABLES와 COLUMNS 파일에서, 전달
            받은 fid에 해당하는 내용을 삭제한다. */

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ 테이블명 변경 프로시저

AlterTable은 oldtname과 newtname을 입력으로 받아서 기존의 테이블명인 oldtname을 새로운 테이블명인 newtname으로 변경하는 함수이다.

```
AlterTable(oldtname, newtname)
    /* oldtname, newtname : IN table name */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

        fid = Search_TableFile(oldtname);
            /* TABLES 파일에서 입력인 oldtname에 해당하는 fid
            를 찾아서 반환한다. */
        Write_TableFile(newtname, fid); /* TABLES 파일에서, 전
            달받은 fid에 해당하는 테이블명을 newtname으로 변
            경한다. */

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ 테이블 속성 변경 프로시저

AttributeTable은 테이블 내에서 변경하고자 하는 컬럼명의 리스트인 oldcolnames를 입력받고, 수정할 새로운 컬럼명을 가지는 (column, type) 쌍과 추가될 (column, type) 쌍의 집합으로 구성된 newcolumns를

입력받아서, 해당 테이블에 대응하는 COLUMNS 파일의 내용을 변경하는 함수이다.

```
AttributeTable(tname, oldcolnames, newcolumns)
    /* tname       : IN table name
       oldcolnames : IN set of column name
       newcolumns  : IN set of (column, type) */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

        fid = Search_TableFile(tname);
            /* TABLES 파일에서 입력으로 들어온 tname에 해당
            하는 fid를 찾아서 반환한다. */
        Read_ColumnFile(fid); /* COLUMNS 파일에서, 전달받은 fid
            에 해당하는 기존의 (column, type) 쌍을 읽어서 모두
            반환한다. */
        Write_ColumnFile(oldcolumns, oldcolnames, newcolumns,
            fid);
            /* oldcolumns : IN set of (column, type),
            Read_ColumnFile( )으로부터 온 기존의 (column, type)
            쌍을 oldcolumns라는 이름으로 전달받아서, oldcolnames
            와 newcolumns를 참조하여 oldcolumns에서 원하는 컬
            럼명을 수정하고 새로운 (column, type) 쌍을 추가하여
            COLUMNS 파일의 해당 fid에 있는 내용을 변경한다.*/

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ Filtering 프로시저

HTMLFiltering 함수는 입력으로 들어온 텍스트 파일을 읽어서 기존의 HTML 태그와 데이터베이스로의 접근에 관련하여 본 논문에서 확장된 HTML 태그를 분리한다. 그리고 확장된 태그의 데이터베이스로의 접근 종류에 따라 각각의 기능을 할 수 있도록 부함수를 호출한다.

```
HTMLFiltering(script)
    /* script : IN text file */
{
        if (tag in HTML tag) next stream;
        else switch (tag) {
case 'JVAR'  : /* 데이터베이스로의 접근에 관련된 검색, 삽입, 삭
               제, 수정 등의 기능에 필요한 변수를 지정한다. */;
case 'JSEL'  : JSEL(tname, condition, columnlist, sortcolumns, result);
case 'JINS'  : JINS(tname, columns);
case 'JDEL'  : JDEL(tname, condition);
case 'JUPD'  : JUPD(tname, columns, condition);
case 'JIF'   : /* 조건에 따라 처리문의 수행 여부를 결정하고 조
               건을 만족하는 경우에 처리문을 단 한 번만 수행한
               다. */;
case 'JWHILE': /* 조건에 따라 처리문의 수행 여부를 결정하며 조
               건을 만족하는 경우에 계속해서 처리문을 반복하여
               수행한다. */;
case 'JERR'  : /* 에러코드번호에 따라 적당한 처리문을 수행한다.
               */;
default      : /* script 파일의 tag 입력 부분의 오류를 검사한다.
               */;
        }
}
```

■ 데이터 검색 프로시저

JSEL 함수는 먼저 입력에 의한 테이블에서 조건인 condition에 해당하는 행을 찾는다. 그리고 컬럼명과 순서의 쌍들로 구성된 sortcolumns를 참조하여 해당 컬럼값의 오름차순 혹은 내림차순으로 찾아낸 행을 재배열한다. 재배열된 행에서, 입력인 columnlist에 의한 컬럼만을 넘겨준다.

```
JSEL(tname, condition, columnlist, sortcolumns, result);
    /*tname      : IN table name
      condition  : IN example ⇒ 'Bcol_1='student' & Bcol_2 >=
                   160 | Bcol_3='A''
      columnlist : IN set of column name
      sortcolumns: IN set of (column, order)
      result     : OUT set of (oid, data) */
{
      sm_Initialize( );
      sm_BeginTransaction(tid);
      sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

      fid = Search_TableFile(tname); /* TABLES 파일을 읽어서 입
            력으로 들어온 테이블명인 tname에 해당하는 fid를 찾
            아서 반환한다. */
      Search_ControlColumn(fid);                    /*
            COLUMNS 파일에서, 전달받은 fid에 해당하는 컬럼
            들을 찾은 후에 컬럼명, 순서, 길이 등에 대한 컬럼의
            모든 정보를 넘긴다. */
      sm_OpenScan(fid, type, groupSize, scanDesc, oid);
            /* fid에 의해 확인된 파일상에서 scan을 초기화한다. */
      while (not end-of-file) {
            sm_ScanNextObject(scanDesc, start, length, retDesc,
            eof);
                  /* 파일 내에서 다음 객체를 읽는다. */
            if (condition) /* 조건에 맞는 컬럼값을 가진 행을 체
                  크한다. 이때 condition에 대한 default값은
                  해당 테이블의 모든 행이다. */
                  if (data not in result)
                        /* 이미 찾아낸 result 내에, 읽어
                        들인 data와 동일한 값이 존재하지
                        않을 경우만 result 속에 새로운
                        data를 추가한다. */
                        add data to result; /* data : set
```

```
                of column value */
        }
        sm_CloseScan(scanDesc); /* scanDesc와 관련된 scan을 close
            한다. */
        if (result is empty) return 02; /* 조건을 만족하는 행이 없을
            경우를 체크한다. */
        Sort(result, sortcolumns); /* result와 (column, order) 쌍의 집
            합인 sortcolumns를 참조하여 해당 컬럼의 정보에 의해
            result를 sort한다. order의 default값은 오름차순이며 내림차
            순의 값이 order에 지정될 수 있다. 여기서 sortcolumns 자
            체에 대한 default 값은 oid 순이다. */
        Check(result, columnlist); /* result 속의 data 중에서 검색하기
            를 원하는 컬럼명의 리스트인 columnlist에 속하는 것으로
            result를 수정한다. 이때 columnlist의 default값은 해당 테이
            블의 모든 컬럼이다. */

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ 데이터 삽입 프로시져

JINS 함수는 먼저, 입력으로 들어온 테이블명을 TABLES 파일에서 찾아서 대응하는 fid를 얻는다. 그리고 TABLES 파일에서 얻은 fid를 이용하여 COLUMNS 파일의 해당 컬럼에 입력값을 삽입한다.

```
JINS(tname, columns)
    /* tname   : IN table name
       columns : IN set of (column, value) */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

        fid = Search_TableFile(tname); /* TABLES 파일에서, 입력
            으로 들어온 tname 테이블명에 해당하는 fid를 찾아서
            반환한다. */
        Search_ColumnFile(columns, fid);
            /* COLUMNS 파일에서, 전달받은 fid에 대응하는 컬
            럼들을 찾은 후에 그 중에서 columns에 해당하는 컬럼
            의 순서, 길이 등에 대한 모든 정보를 넘긴다. */
        if (key column is not included) return 01;
            /* 기본키인 컬럼에 값이 입력되지 않았을 경우를 체크
            한다. */
        e = JSEL(tname, condition, columnlist, sortcolumns, result);
            /* condition : key column = value */
        if (e != 02) return 01; /* 기본키인 컬럼에 중복된 값이 입력
                된 경우를 체크한다. */
        sm_CreateObject(groupIndex, fid, nearHint nearObj, objHdr,
            length, data, oid);
            /* 전달받은 fid와 Search_ColumnFile 함수의 정보를
            참조하여 해당 컬럼에 입력으로 들어온 값을 삽입한다.
            즉 fid에 의해 확인된 파일 내에 객체를 생성한다. 그리
            고 값이 입력되지 않은 컬럼에는 null값을 채운다. */

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ 데이터 삭제 프로시져

JDEL는 해당 테이블에서 입력으로 들어온 조건이 만족되는 행을 삭제하는 함수이다. 만약 조건이 입력되지 않은 경우에는 해당 테이블의 모든 행이 선택되어 삭제된다.

```
JDEL(tname, condition)
    /* tname     : IN table name
       condition : IN example ⇒ ' Acol_1=3 & Acol_2 != 'F' ' */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

        JSEL(tname, condition, columnlist, sortcolumns, result);
            /* tname 테이블에서 입력된 조건에 맞는 행을 찾아서
            넘긴다. */
        for (oid in result) /* JSEL 함수에서 전달받은 result 속에 데
            이터가 존재하는 동안 sm_DestroyObject( )를 실행한
            다. */
            sm_DestroyObject(groupIndex, oid); /* oid에 의해서
            확인된 객체를 삭제한다. */

        sm_CloseBufferGroup(groupIndex);
        sm_CommitTransaction(tid);
        sm_ShutDown( );
}
```

■ 데이터 수정 프로시져

JUPD는 해당 테이블에서 입력으로 들어온 조건이 만족되는 행을 찾은 후에, 테이블에 존재하는 컬럼명과 변경하고자 하는 새로운 컬럼값의 쌍들로 구성된 columns를 참조하여 해당 컬럼의 값을 수정하는 함수이다.

```
JUPD(tname, columns, condition)
    /* tname     : IN table name
       columns   : IN set of (column, value)
       condition : IN example ⇒ ' Qcol_1 >= 150 & Qcol_2 =
                   'seoul' ' */
{
        sm_Initialize( );
        sm_BeginTransaction(tid);
        sm_OpenBufferGroup(groupSize, policy, groupIndex, flags);

        fid = Search_TableFile(tname);
            /* TABLES 파일에서 입력인 tname에 해당하는 fid를
```

```
        찾아서 반환한다. *
    Search_ColumnFile(columns, fid);
        /* COLUMNS 파일에서, 전달받은 fid에 대응하는 컬
        럼들을 찾은 후에 그 중에서 columns에 해당하는 컬럼
        의 순서, 길이 등에 대한 모든 정보를 넘긴다. *
    JSEL(tname, condition, columnlist, sortcolumns, result); /*
        읽어들일 tname 테이블에서 condition 조건에 맞는 행
        을 찾아서 원하는 컬럼의 값을 oid와 함께 넘겨준다.
        */
    for (oid in result)
        /* JSEL 함수에서 전달받은 result 속에 데이터가 존재
        하는 동안 입력인 columns와 Search_ColumnFile 함수
        의 정보를 참조하여 sm_WriteObject( )를 실행한다. */
    sm_WriteObject(groupIndex, start, length, data, userDesc,
        release);
        /* 지정된 데이터를 가지고 해당하는 byte들의 영역을
        수정한다. */
    sm_CloseBufferGroup(groupIndex);
    sm_CommitTransaction(tid);
    sm_ShutDown( );
}
```

5. 결 론

본 논문에서는 인터넷을 위한 WWW 데이터베이스 인터페이스로써 UCM 시스템을 제안하였다. UCM 시스템은 기존의 CGI 실행 방식에서 클라이언트로부터 요청이 많을 경우에 발생하는 시스템의 성능 저하 문제를, 데몬으로 운영되는 통합된 CGI 프로그램인 JCGId를 사용함으로써 개선할 수 있다. 즉 클라이언트의 요구 종류에 따라 계속 다른 CGI를 생성하는 것이 아니라 데몬 방식을 사용하는 통합된 하나의 CGI를 사용함으로써 다양한 요구들에 대한 서비스를 통합적으로 해준다. 또한 본 논문에서 제안한 시스템은 DBMS를 이용하지 않고 데이터베이스로 접근할 수 있도록 한다. 이러한 UCM 시스템은 사용자에게 웹 인터페이스를 제공하는 간단한 데이터처리기를 포함하기 때문에 일반 인터넷 사용자 입장에서 볼 때 WWW 데이터베이스를 더 경제적으로 이용할 수 있다는 장점이 있다. 이 사실을 좀 더 구체적으로 살펴보면, UCM 시스템에서는 사용자와 데이터베이스와의 연계가 확장된 HTML 태그를 통하여 이루어 지는데 이때 기존의 연동 구조에서 사용되는 SQL 언어가 이용되지 않는다. 그리고 CGI와 데이터베이스를 연결하는 인터페이스로는 EXODUS 저장 관리자를 사용하였다.

한편 본 논문에서 제안된 시스템은 데이터베이스 접근에 대하여 가장 핵심적이며 제한적인 기능만을 가진다. 즉 본 시스템은 기존의 상용화된 DBMS에서 가지는 두 개 이상의 테이블에 대한 조인(join) 등 여러 가지 기능과 다양한 사용자 인터페이스, 그리고 보안 유지에 대한 부분 등에서 보완되어야 할 몇 가지 문제점을 갖고 있다. 하지만 UCM 시스템은 확장이 가능한 구조이므로 새로운 태그 및 속성을 추가함으로써 더 다양한 기능들을 포함할 수 있다. 그리고 SQL 언어 대신 확장된 태그를 이용하는 점은 DBMS를 사용하지 않는다는 면에서 큰 장점이 될 수 있는 반면에 SQL 언어에 익숙해 있던 사용자들 입장에서는 새로운 태그를 익혀야 하는 부담이 있게 된다. 따라서 이러한 단점을 보완하기 위한 방법으로, 사용자가 HTML 문서를 데이터베이스 안에 저장, 관리할 수 있는 웹 페이지를 개발하여 더욱 편리한 데이터베이스 접근 환경을 구축할 예정이다.

참 고 문 헌

[1] P. C. kim, "A Taxonomy on the Architecture of Database Gateways for the Web," Proc. of the 13th ICAST and 2nd ICMIS, Shaumburg, pp.226-232, 1997.

[2] M. J. Carey, D. J. Dewitt, J. E. Richardson, and E. J. Shekita, "Object and File Management in the EXODUS Extensible Database System," Proc. of 12th Int. Conf. on Very Large Data Bases, pp.91-100, Aug. 1986.

[3] ftp://ftp.cs.wisc.edu/exodus/sm/, EXODUS Storage Manager Version 3.1 Documentation, Nov. 1993.

[4] The Illustra DataBlade ClSurpporting Documentation, Informix, 1996.

[5] UniSQL-UniWeb, Kcom, 1996.

[6] Debby Kramer, Introduction to Oracle: SQL and PL/SQL, Vol.1, Vol.2, Vol.3, Oracle Corporation, 1996.

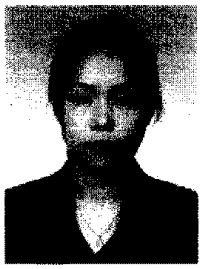
[7] Web Application Server 3.0 Overview, Oracle, 1997.

[8] http://home.netscape.com/newsref/std/server_api.html, Netscape Server API.

[9] http://www.microsoft.com/win32dev/apiext/isapimrg.htm, Internet Server API.

[10] David Chappell, Chappell and Associates, "Dis-

tributed Object Computing With CORBA." Networld+Interop'94, Zoff-Davis Exposition and Conference Company, 1994.



김 은 경

e-mail : jlikek@songsim.cuk.ac.kr

1993년 가톨릭대학교 수학과 졸업(학사)

1998년 가톨릭대학교 대학원 수학과(전산학전공) 졸업(이학석사)

1998년~현재 가톨릭대학교 컴퓨터공학과 시간강사

관심분야 : WWW 데이터베이스, 공간 데이터베이스(GIS), 객체지향 데이터베이스

황 병 연

e-mail : byhwang@dbs.cuk.ac.kr

1986년 서울대학교 공과대학 컴퓨터공학과 졸업(학사)

1989년 한국과학기술원 전산학과 졸업(공학석사)

1994년 한국과학기술원 전산학과 졸업(공학박사)

1994년~현재 가톨릭대학교 컴퓨터공학과 부교수

관심분야 : 공간 데이터베이스(GIS), WWW 데이터베이스, 전자상거래