Z39.50 서버의 설계 및 CORBA를 이용한 Z39.50 서버와 데이터베이스 엔진의 통합

손 충 범[†] · 유 재 수[††]

요 약

CORBA는 분산환경에서 이기종 간의 시스템을 통합하는 방법을 제시하고 있다. 기존에 Z39.50 프로토콜을 지원하는 서버들이 많이 개발되어 현재 도서관, 기업 등에서 사용 중에 있으며, 이런 서버들은 저마다의 데이터베이스를 구축하여 정보 서비스를 제공하고 있다. 본 논문에서는 기존의 서버들보다 다양한 Z39.50 서비스들을 지원하는 Z39.50 서버를 설계 및 구현한다. 또한 CORBA를 이용하여 다양한 데이터베이스 엔진들과 구현된 Z39.50 서버를 통합한다. 구현한 Z39.50 서버는 기본적인 서비스인 접속, 탐색, 종료 서비스를 제공하고, 검색어를 조회하는 스캔서비스, 대용량의 레코드들을 전송하기 위한 분할서비스, 서버의 구현 정보를 설명하는 설명 기능을 지원한다.

Design of a Z39.50 Server, and Integration of the Z39.50 Server and Database Engines using COBRA

Chung-Beom Son[†] · Jae-Soo Yoo[††]

ABSTRACT

The CORBA presents the method of integrating heterogeneous systems in the distributed environment. In recent, many existing information retrieval servers with Z39.50 protocol have been developed and used in the libraries and companies, etc. The servers construct each database and provide users with various information services. In this paper, we design and implement a Z39.50 server that supports various Z39.50 protocol services over the existing servers. We also integrate various database engines and the Z39.50 server using CORBA. Our Z39.50 server basically provides the init service, the search service, and the close service. In addition, it supports the scan service browsing a term list, the segment service presenting large records, and the explain facility explaining the implementation information of the server.

1. 서 론

현대 사회는 다양한 형태로 존재하는 수많은 정보들을 접하고 활용하는 시대이다. 사용자들은 이런 다양한 형태의 정보들을 제공하는 정보검색 시스템에 접근하여 원하는 정보를 구할 수 있는데, 각 시스템마다 서로 다른 형태의 탐색 구문과 사용자 인터페이스를 제공하고 있어서, 사용자 측면에서 각 시스템의 서로 다른 탐색 구문과 사용자 인터페이스를 배워야 하는 어려움을 느끼고 있다. 이러한 문제점을 해결하고 이기종간의 상호 운용성을 보장하기 위해 미국 국가 표준인 Z39.50 프로토콜이 제정되었다[1]. 이 프로토콜은 분산 네트워크 환경에서 시스템 간의 정보검색을 위해 서로 통신하는 표준화된 방법을 정의하고 있어서, Z39.50 프로토콜을 이용할 경우 정보 검색 절차와 방법이 표준화가 되기 때문에 기존의 정보검색 시스템에서 여러 개의 데이터베이스를 검색하는 데에 따른 많

※ 본 연구는 한국과학재단 특정기초연구(과제번호 : 1999-1-303-077-3)의 지원에 의해서 부분적으로 수행되었음.
† 준 회 원 : 충북대학교 대학원 정보통신공학과
†† 종신회원 : 충북대학교 전기전자공학부 교수
논문접수 : 2000년 3월 28일, 심사완료 : 2000년 11월 24일

은 문제점들을 해결할 수 있다. 하지만 Z39.50 프로토콜을 지원하는 기존의 정보검색 시스템은 새로운 데이터베이스 엔진 모듈에 검색엔진, 관계형 데이터베이스, 객체지향 데이터베이스 등을 추가할 경우 개발비용이 많이 들고 사용자가 많아질수록 서버의 처리 부담이 커지는 문제점이 있다.

CORBA(Common Object Request Broker Architecture)[13]를 이용하여 시스템을 구현할 경우의 장점으로는 네트워크 프로그램을 할 필요가 없어 개발 시간이 단축되고, 모듈이 언제든지 확장, 수정 가능하므로 유지 보수가 쉽다. 또한, 기존 시스템을 수정하지 않고 기존 모든 자원을 재사용할 수 있다.

본 논문에서는 분산되어 있는 Z39.50 서버의 데이터베이스 엔진들을 CORBA를 이용하여 통합하는 방법을 제시하고 또한 CORBA를 이용한 Z39.50 서버를 설계하고 구현한다. 구현한 Z39.50 서버는 기본적인 서비스인 접속, 탐색, 종료 서비스를 제공하고, 검색어를 조회하는 스캔서비스, 대용량의 레코드들을 전송하기 위한 분할서비스, 서버의 구현 정보를 설명하는 설명 기능을 지원한다. 이처럼 CORBA를 이용함으로써 Z39.50 서버의 처리 부담을 줄이고, 이질적인 검색엔진, 관계형 데이터베이스, 객체지향 데이터베이스를 Z39.50 서버와 쉽게 통합할 수 있어 시스템 확장에 있어 많은 편리함을 제공한다. 만약 제안한 구조로 Z39.50 서버를 구현할 경우 타 시스템에 있는 데이터베이스 엔진 모듈 부분을 그대로 사용할 수 있어 자원 재사용성을 높일 수 있다.

본 논문의 구성은 2장에서 Z39.50 정보검색 프로토콜과 CORBA에 대해 살펴보고, 3장에서 구현한 시스템 구성도와 설계한 IDL(Interface Definition Language)에 대해 설명하고 4장에서는 구현한 시스템 환경과 구현한 Z39.50 기능과 서비스들에 대해 설명한다. 마지막으로 5장에서는 결론을 맺는다.

2. 관련 연구

2.1 Z39.50 정보검색 프로토콜

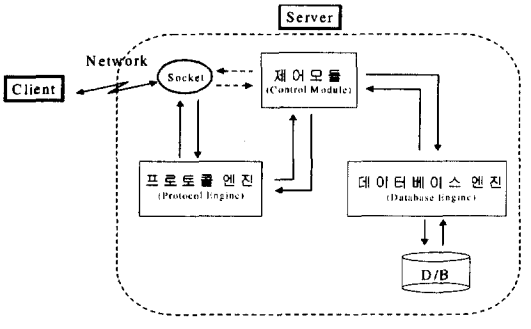
Z39.50 정보검색 프로토콜은 정보 검색을 목적으로 한 두 컴퓨터간의 표준화된 통신 규약을 규정하고 있다. 이 프로토콜을 이용할 경우 정보 검색의 절차와 방법이 표준화되기 때문에 대규모 정보 데이터베이스 또는 정보 자원을 보다 쉽게 이용할 수 있다. Z39.50 프로토콜의 일반적인 특징은 두 대의 컴퓨터가 서로 상호작용하기 위해 사용하는 포맷과 절차들을 다루는 일련의 규칙들의 집합인 네트워크 프로토콜로서 세션 지향적이고 연속적이다. 또한 네트워크 정보 검색용 표준 프로토콜이고 클라이언트/서버 모형을 기본으로 하고 있다. Z39.50 프로토콜의 서비스 유형에는 메시지 요구와 응답으로 이루어진 확인 서비스와 요구만 있고 응답이 없는 비확인 서비스, 확인 서비스와 비확인 서비스를 모두 사용할 수 있는 제한적 확인 서비스 등이 있다. Z39.50의 기본적인 기능들은 접속, 탐색, 검색, 종료기능이 있고, 이 외에도 결과 집합 삭제, 브라우즈, 정렬, 접근 제어, 자원 제어, 설명, 확장 서비스 기능이

〈표 1〉 Z39.50 기능 설명

기 능	서비스	설 명
접속 기능	접속 서비스	서버와 접속하기 위하여 클라이언트 요구
탐색 기능	탐색 서비스	검색하여 서버에 있는 레코드들에 대한 결과집합을 만들기 위한 클라이언트 요구
검색 기능	전송 서비스	검색된 결과집합으로부터 원하는 레코드를 전송 받기 위한 클라이언트 요구
	분할 서비스	검색된 결과집합으로부터 원하는 레코드들을 서버가 한번에 제공하기 위한 서버 요구
결과집합삭제 기능	삭제 서비스	검색된 결과집합을 삭제하기 위한 클라이언트 요구
브라우즈 기능	스캔 서비스	인덱스 용어에 대한 클라이언트 요구
정렬 기능	정렬 서비스	결과집합을 정렬하기 위한 클라이언트 요구
접근제어 기능	접근제어 서비스	클라이언트 요구를 중지시킨 후에 인증 정보에 대한 서버의 요구
자원관리 기능	자원관리 서비스	클라이언트 요구를 중지시킨 후에 현재 또는 진행된 자원사용의 확인에 대한 서버의 요구
	트리거자원제어 서비스	서버가 현재 연산을 취소하거나 자원제어요구가 클라이언트에게 보내질 것을 요구하기 위한 클라이언트 요구
	자원보고 서비스	자원 보고에 대한 클라이언트 요구
설명 기능		서버의 구현 정보를 제공받을 수 있기 위한 클라이언트 요구
확장서비스 기능	확장서비스 서비스	프로토콜 이외의 서비스들에 대한 접근을 제공하기 위한 클라이언트 요구
해제 기능	종료 서비스	세션을 중지하기 위하여 클라이언트 또는 서버 요구

제공된다. <표 1>은 각 기능에 대한 설명이다.

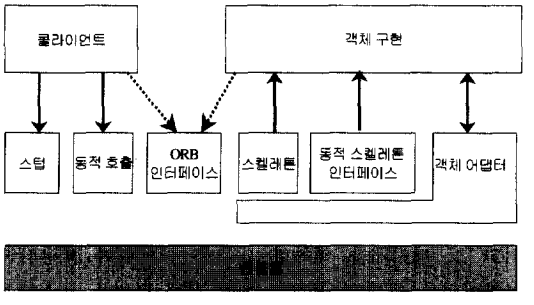
일반적인 Z39.50 서버 모듈은 (그림 1)에서 보는 바와 같이 제어 모듈, 프로토콜 엔진, 데이터베이스 엔진으로 구성된다. 프로토콜 엔진은 PDU(Protocol Data Unit)라 불리는 Z39.50 네트워크 메시지를 읽거나 쓰는 작업을 하는 모듈이다. 제어 모듈은 입력 이벤트 발생시 프로토콜 엔진을 호출하여 PDU를 읽고 그 PDU를 해독한 내용을 데이터베이스 엔진에 전달한다. 데이터베이스 엔진이 이를 처리하여 제어모듈에 넘겨주면, 프로토콜 엔진을 호출하여 인코딩한 후 네트워크에 쓰게 된다. 데이터베이스 엔진은 질의를 수행하고, 검색된 결과를 저장한다[8].



(그림 1) 기본적인 Z39.50 서버의 구성

2.2 CORBA

오늘날의 컴퓨팅 환경은 수많은 소프트웨어, 하드웨어 플랫폼이 네트워크 상에 분산되어 있는 복잡한 구조를 갖는다. 따라서 정보 자원의 통합이라는 시스템 통합의 관점에서 네트워크 상에 분산되어 있는 다양한 종류의 정보들을 어떠한 방식으로 통합 관리할 것인가가 중요하게 되었다. CORBA란 이러한 애플리케이션들이 어디에 위치하고, 어떤 언어로 구현되었든지 관계없이 다양한 플랫폼과 운영 시스템을 지원함으로써 오늘날 분산 네트워크환경을 지원하는 새로운 시스템 통합기술이다. CORBA는 주로 ORB(Object Request Broker)부분에 대한 객체지향 구조의 표준을 정의한 것으로써 분산 객체간의 상호 운용을 위한 통신 미들웨어 역할을 하는 버스구조로 정의할 수 있으며, 객체지향 컴퓨팅을 기반으로 하기 때문에 추상화(Abstraction), 캡슐화(Encapsulation), 상속(Inheritance), 다형성(Polymorphism) 등을 제공함으로써 소프트웨어의 재사용성 등을 높임으로써 응용 프로그램 개발을 용이하게 해준다. 또한 분산 환경에서 클라이언트와 서버간의 인터페이스만 정의하면 이들 서로간의 서비스 요구나 결과 값의 전달이 하부 통신 메카니즘에 의해서 투명하게 이루어진다. (그림 2)는 CORBA의 서버와 클라이언트 구조를 나타내고 있다. 클라이언트는 요구를 하기 위해 동적 호출이나 정적 스텁을 사용할 수 있다. 클라이언트는 또한 ORB 인터페이스를 통하여 ORB와 직접 상호 교섭할 수도 있다. 클라이언트가 스텁을 통해서 요구를 하면 스텁은 프록시 객체를 제공하여 클라이언트가 메소드를 호출할 수 있게 해준다. 마샬된 메소드 호출정보는 ORB를 통해 스켈레톤에 전달된다. 스켈레톤은 클라이언트로부터 요구된 마샬된 메소드 호출정보를 언마샬하고 원하는 객체의 해당 메소드들을 찾아 객체 구현에서 일을 처리할 수 있도록 해준다[4].



(그림 2) CORBA의 구조

3. CORBA를 이용한 Z39.50 서버의 설계

3.1 CORBA를 이용한 Z39.50 서버의 통합 방안

3.1.1 객체 랩핑 방법

객체 랩핑 기술은 기존에 구축된 시스템을 서로 통합하는 방법이다. 이런 기존에 구축된 시스템은 대개 복잡한 객체들을 포함하고 있다. 즉, 객체 랩핑 기술은 이런 복잡한 객체들을 관리하기 위한 주요 기술을 제공한다. 개발자들은 레거시 시스템에 대해 제한된 제어를 하게 된다. 하지만, 객체 랩퍼는 제어할 수 있는 형태로 시스템을 재구성한다. 객체 랩퍼는 또한 시스템이 적응 가능하도록 복잡한 부시스템들 사이에 분리를 제공한다[12]. 만약 이 방법을 사용하여 시스템을 통합하는 경우, 기존에 개발된 Z39.50 클라이언트 프로그램들을 사용할 수 없는 단점이 있다.

3.1.2 서버 확장 방법

서버 확장 방법은 기존에 개발된 Z39.50 서버의 구조를 그대로 활용한다. Z39.50 서버의 구조를 보면, 제어모듈, 프로토콜 엔진모듈, 데이터베이스 엔진모듈로 구성되어 있다. 이 구조에서 데이터베이스 엔진 모듈을 서버와 분리하여 CORBA로 통합하는 것이다. 이렇게 함으로써 기존에 개발된 Z39.50 서버들의 데이터베이스 엔진모듈들을 쉽게 통합할 수 있어 시스템 확장이 용이하다. 또한 기존에 개발된 Z39.50 클라이언트 소프트웨어들을 그대로 사용할 수 있다. 본 논문에서는 CORBA를 이용하여 서버를 확장하는 방법으로 Z39.50 서버를 구현하였다.

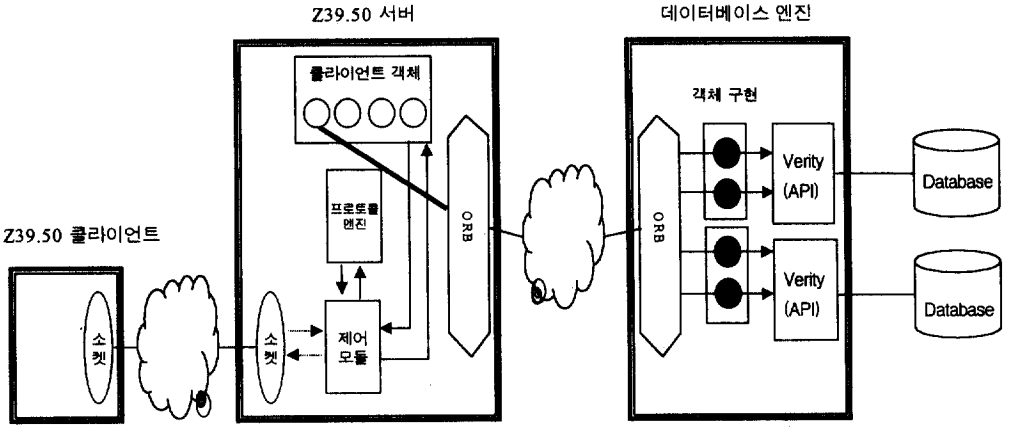
3.2 시스템 구성도

본 논문에서 구현한 Z39.50 서버의 전체적인 시스템 구성도는 (그림 3)과 같다. 일반적인 Z39.50 서버는 제어 모듈, 프로토콜 엔진 모듈, 데이터베이스 엔진 모듈로 구성되어 있다. 구현한 서버는 그림에서 보듯이 앞에서 언급했던 Z39.50 서버 통합 방법 중에 두 번째 서버 확장 방법을 이용하여 Z39.50 서버에서 데이터베이스 엔진 모듈을 분리하여 CORBA를 이용하여 구현하였다. 구현한 시스템의 처리 과정을 보면 입력 이벤트 발생시 프로토콜 엔진을 호출하여 메시지(PDU)를 읽고 그 메시지를 해독한 내용을 분석하여 해당되는 클라이언트 객체가 ORB를 통해 객체 구현을 호출하게 된다. 객체 구현에서는 ORB를 통해 Z39.50 서버의 요구를 받아 처리할 적절한 객체를 호출하게 되는데 상용검색엔진인 Verity의 API을 이용하는 객체나 객체지향 데이터베이스인 O_2 API를 이용하는 객체를 호출하여 처리되고, 처리된 결과를 ORB를 통해 제어 모듈로 넘겨주면, 제어 모듈에서 프로토콜 엔진을 호출하여 인코딩한 후 소켓을 통해 전송하게 된다. 이처럼 데이터베이스 엔진 부분을 서버와 분리하여 이곳에 CORBA를 이용하여 통합함으로써 미들웨어 역할을 담당하게 된다.

3.3 IDL 정의

본 논문에서 구현한 Z39.50 서버는 Z39.50 서비스들 중에 접속, 탐색, 전송, 분할, 종료서비스를 지원하고, 부가적으로 검색어를 조회하는 스캔서비스. 서버의 구현정보를 설명하는 설명기능을 제공한다. (그림 4)는 Z39.50 프로토콜을 지원하는 서버가 CORBA 환경에서 이런 서비스를 지원할 수 있도록 인터페이스 정의어인 IDL을 사용하여 각 서비스들을 정의하였다.

(그림 4-a), (그림 4-b)의 설계한 IDL을 보면 네 개의 구조체와 두 개의 인터페이스로 구성되어 있다. 구조체 search_result는 탐색 서비스를 수행한 후에 검색결과를 반환하기 위한 것이고, present_result는 Z39.50 클라이언트가 전송 서비스를 요구할 경우 검색결과들을 전송하기 위해 사용된다. 또한 scan_request는 스캔서비스 요구시 필요한 매개변수를 정의한 구조체이고, scan_result는 스캔서비스 요구에 대한 결과를 반환하기 위한 구조체이다. Verity 인터페이스의 verity_init() 메소드는 Z39.50 클라이언트가 접속 요구를 하면 이 메소드가 호출되어 세션이 설정되고, 설정이 완료된 후 사용자가 검색하기 위해 탐색서비스를 요구하면, verity_search() 메소드는 사용자가 질의한 내용에 대해 탐색을 수행한다. 사용자는 검색결과 중에서 원하



(그림 3) 구현한 시스템의 전체적인 구성도

```
struct search_result { /* 탐색서비스의 결과를 유지하는 구조체 */
  short hit;
  short errcode;
  string errstring;
};
struct present_result { /* 요구된 레코드의 전송을 위한 구조체 */
  short length;
  string record;
  short errcode;
  string errstring;
};
struct scan_request    /* 스캔서비스 요구를 위한 구조체 */
{
  short num_bases;
  string basenames;
  string attributeset;
  string term;
  short term_position;
  short num_entries;
};
struct scan_result {    /* 스캔 결과를 전송하기 위한 구조체 */
  string term;
  short occurrences;
  short term_position;
  string status;
  short errcode;
  string errstring;
};
```

(그림 4(a)) 구조체들

```
interface Verity {
  void verity_init();        /* 세션을 설정하는 메소드 */
  search_result verity_search(in string basename, in string query);
                             /* 탐색을 수행하는 메소드 */
  present_result verity_present(in short number, in short element,
  in short flag);            /* 레코드를 전송받기 위한 메소드 */
  scan_result verity_scan(in scan_request query);
                             /* 검색어를 조회하기 위한 메소드 */
  void verity_close();       /* 세션을 종료하는 메소드 */
};
interface O2 {
  void o2_init();          /* 세션을 설정하는 메소드 */
  search_result o2_search(in string basename, in string query);
                             /* 탐색을 수행하는 메소드 */
  present_result o2_present(in short number, in short element, in
  short flag);               /* 레코드를 전송받기 위한 메소드 */
  scan_result o2_scan(in scan_request query);
                             /* 검색어를 조회하기 위한 메소드 */
  void o2_close();       /* 세션을 종료하는 메소드 */
};
```

(그림 4(b)) 정보검색을 위한 IDL 정의

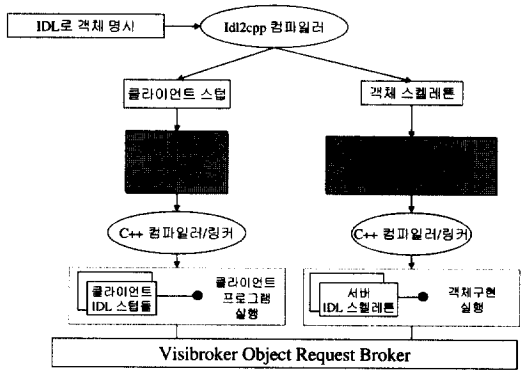
는 레코드들을 요구할 경우 verity_present() 메소드을 사용하여 레코드들을 전송한다. verity_close() 메소드는 세션을 종료하는 기능을 담당한다. O2 인터페이스의 메소드들의 기능은 Verity 인터페이스의 메소드들의 기능과 같다.

본 논문에서 설계한 IDL에 다른 데이터베이스 엔진 모듈을 추가할 경우에 그 모듈에 해당하는 인터페이스를 정의하면 된다.

4. CORBA를 이용한 Z39.50 서버의 구현

4.1 구현 환경

구현한 정보검색 시스템은 Indexdata사의 YAZ[11] 개발툴을 이용하여 데이터베이스 엔진모듈 구현하였고, 추가된 분할기능 및 설명기능에 대한 제어 모듈과 프로토콜 엔진 모듈 부분을 구현하였다. 구현 환경은 Sun SPARC station 5 기종에 운영체제는 Solaris 2.5.1을 사용하였다. 구현 언어로는 C++ 언어를 사용하여 구현하였다. 또한 CORBA 제품으로는 Inprise의 VisiBroker 3.3 for C++[5]를 사용하였다. 데이터베이스 엔진 모듈에 사용한 제품은 상용 검색엔진인 Verity[10]와 객체지향 데이터베이스인 O2[14]이다. (그림 5)는 Visibroker를 가지고 구현한 시스템 구현 절차에 관한 순서도다. (그림 5)에서 보듯이 Z39.50 서버 프로그램 코드에 CORBA 클라이언트 프로그램 코드를 추가하여 Z39.50 서버를 구현하였고, 객체 구현은 데이터베이스 엔진 모듈부분에 검색엔진인 Verity의 API와 객체지향 데이터베이스인 O2 API를 이용하여 구현하였다.



(그림 5) 시스템 구현 절차

4.2 Z39.50 기능 및 서비스들

4.2.1 접속서비스

이 서비스는 클라이언트와 서버 간에 연결을 설정하

고 Z39.50 세션(Z-association)을 설정한다. 즉, 네트워크를 설정하고 Z-association을 초기화한다. Z39.50 세션이 설정될 때, 서버와 클라이언트는 전송될 레코드의 최대 크기, 사용 가능한 선택사항(init, search, present, level 1 segmentation, level 2 segmentation, scan, sort, delete) 등을 결정한다. 이 부분의 구현에는 데이터베이스 엔진 모듈을 초기화하는 함수들이 사용된다.

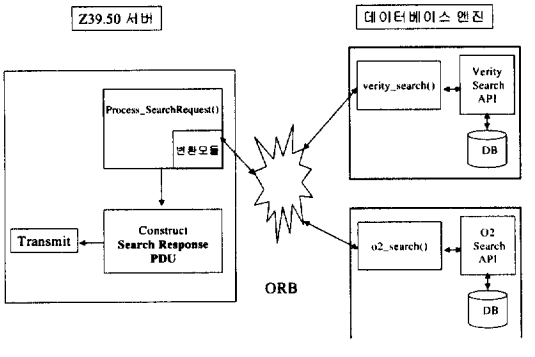
4.2.2 탐색서비스

탐색서비스는 클라이언트가 탐색하고자하는 질의어를 사용하여 서버에 탐색 요구를 하는 기능이다. 본 논문에서 구현한 Z39.50 서버가 제공하는 탐색 서비스의 특징은 다음과 같다.

〈표 2〉 탐색서비스 특징

질의 형태	type - 1, type - 101 (RPN)
애트리뷰트 집합	Bib-1, Exp-1
Types of Bib-1	Use, Relation, Structure, Truncation
Types of Exp-1	Use, Imports some Bib-1 types

<표 2>에서 Type-1, Type-101 질의 형태는 RPN (Reverse Polish Notation)형태인 반면 Verity에서 제공하는 질의 형태는 중위(infix), 후위(postfix) 또는 전위(prefix)의 문자열 질의 형태이다. O2 데이터베이스에서는 RPN 형태를 OQL 질의어로 변환하는 모듈이 추가된다. 또한 구축하고자하는 데이터베이스의 애트리뷰트들과 Bib-1에 정의된 애트리뷰트들도 서로 다르다. 따라서 상호간에 매핑시켜 줄 수 있는 모듈이 추가되어야 한다. 질의 변환 모듈에서 Verity 검색엔진에 사용되는 질의 형태로 변환하여 생성된 질의를 가지고 verity_search() 함수를 호출한다. verity_search() 함수에서는 Verity Search API를 호출하고 탐색을 수행하여 결과 집합을 생성한다. (그림 6)은 구현한 서버의 탐색서비스의 구현 모듈이다.



(그림 6) 탐색서비스 구현모듈

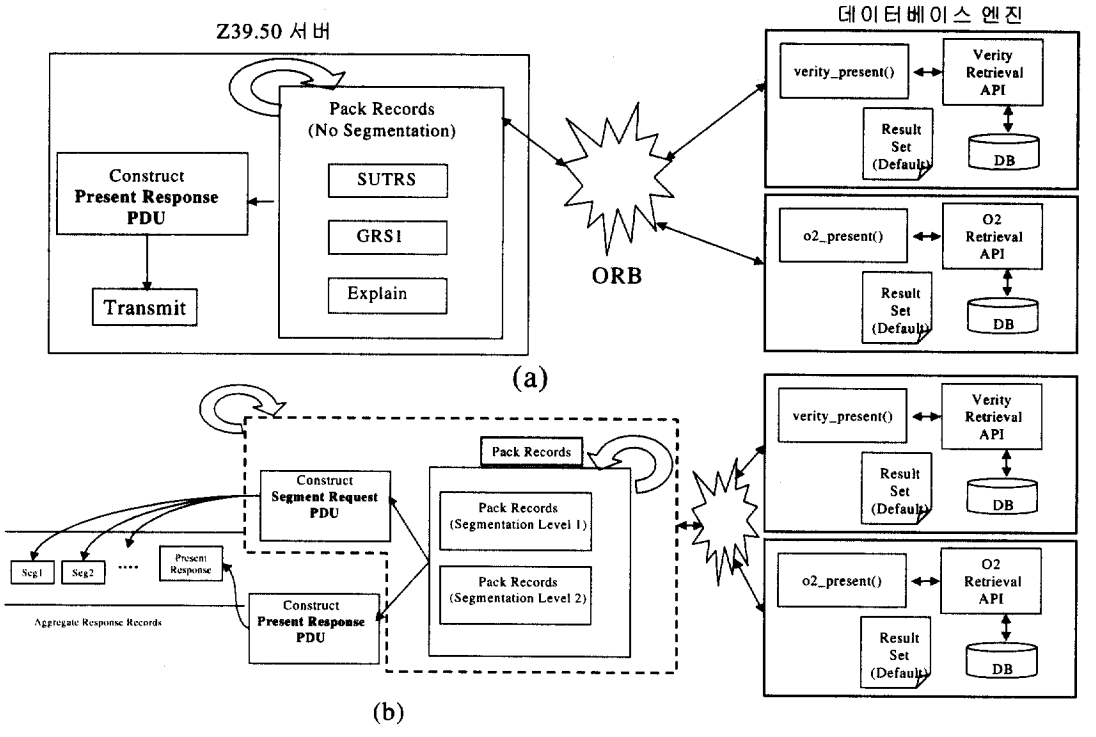
4.2.3 전송 및 분할서비스

전송 서비스는 사용자가 원하는 레코드들을 요구하는 기능이고 분할 서비스는 클라이언트가 요구하는 레코드들의 전체 크기가 클라이언트와 서버사이에 약속한 크기보다 클 경우 이 레코드들을 한번에 효율적으로 전송할 수 있도록 레코드들을 여러 세그먼트로 나누어 연속적으로 전송하고자 하는 기능이다. 구현한 서버에서 지원하는 검색 기능의 특징은 <표 3>과 같다. 현재 지원되는 레코드 구문은 SUTRS와 Explain이다. Explain 레코드 구문에 대한 설명은 설명 기능 구현 부분에서 제시된다. 전송 요구를 할 때 엘리먼트 집합 "B" 와 "F"의 구성요소 정의는 데이터베이스마다 다르다. 일반적으로 클라이언트가 "F"로 지정하면 모든 레코드를 전송하고, "B"로 지정하면 레코드의 요약정보에 해당하는 부분을 전송한다. 본 논문에서는 레코드 구성요소 중 일부인 Title, Date, Author, Subject만을 전송한다.

〈표 3〉 검색기능 특징

레코드 구문	SUTRS, Explain
엘리먼트 집합	B(Brief), F(Full)
분할 레벨	Level 1, Level 2

(그림 7)은 구현한 레코드 전송모듈을 나타내고 있다. 분할이 적용되지 않는 경우의 레코드 전송 절차를 (a)에 나타나 있다. 전송하고자하는 레코드들의 전체 크기가 서버와 클라이언트 사이에 약속된 크기보다 클 경우에 나머지 레코드들은 다음 전송 응답을 통해 전송해야 한다. 전송 절차를 보면, Z39.50 서버의 Pack_Records 함수에서 verity_present() 함수를 호출하면 ORB를 통해 데이터베이스 엔진모듈의 Verity Retrieval 함수를 통해 결과 집합(Default)으로부터 해당 범위의 레코드를 하나씩 Pack_Records로 넘겨준다. 넘겨주기 전에 해당 레코드 구문으로 레코드를 구성하는데, 구조화되지 않은 텍스트로 구성하는 SUTRS, 데이터베이스 레코드를 표현하는 GRS-1, 그리고, 설명 데이터베이스의 레코드들을 전송하기 위한 레코드 구문인 Explain 함수를 거친다. 이렇게 레코드 구문에 맞게 구성된 레코드를 넘겨받은 함수 Pack_Records는 프로토콜에 정의된 절차를 통해 전송할 레코드를 만들어낸



(그림 7) 레코드 전송 모듈

다. 전송할 레코드를 만들어 내면 이를 BER 루틴을 통해 인코딩하고 소켓을 통해 전송한다.

분할이 적용될 경우의 레코드 전송 절차는 (b)에 설명되어 있다. 분할 레벨 2에서는 하나의 레코드 자체를 분할할 수 있다. 즉, 레벨 1 분할과 레벨 2 분할의 차이점은 레벨 2 분할은 한 세그먼트보다 큰 레코드에 대해 분할하여 전송할 수 있으나, 레벨 1 분할은 그런 기능을 제공하지 못한다. 결과 집합으로부터 레코드를 페치하는 함수 Backend_fetch는 (a)의 전송절차와 동일하다. Backend_fetch로 읽어 온 레코드들을 가지고 프로토콜에 정의된 레벨 1과 레벨 2의 절차에 따라 전송할 레코드로 구성한다.

이때 서버가 전송하는 PDU들은 세그먼트 요구들과 전송 응답의 조합이다. 제일 마지막에 전송하는 PDU만 전송 응답이고 나머지는 세그먼트 요구이다.

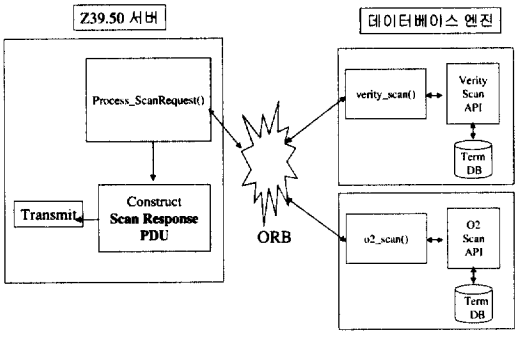
4.2.4 스캔서비스

스캔 서비스는 클라이언트가 원하는 데이터베이스의 정렬된 용어목록을 조회하는 서비스이다. 클라이언트는 조회할 데이터베이스 이름과 용어 목록의 시작점, 끝점을 포함하는 스캔 요구(Scan request)를 서버에게 보내면 서버는 용어 목록의 엔트리 개수와 엔트리들을 클라이언트에 전송해 준다. 이 서비스는 데이터베이스의 용어 목록만 만들어 낼 수 있으면 쉽게 구현할 수 있다. 구현 방법으로는 세 가지를 생각해 볼 수 있다. 첫 번째 방법은 클라이언트가 스캔 요구를 할 때 용어 목록을 만드는 방법이고, 두 번째 방법은 용어 목록을 미리 만들어 화일로 저장해 놓고 클라이언트가 스캔 요구를 할 때 이 화일을 읽어서 응답해주는 방법이다. 마지막 방법은 앞의 두 가지 방법을 혼용하는 방법으로 데이터베이스를 만들 때 레코드들에 대한 용어 목록을 화일로 만드는 방법이다. 첫 번째 방법은 데이터베이스의 변경된 내용을 용어 목록에 동적으로 반영할 수 있는 장점이 있지만 서비스 속도가 느리다는 단점이 있다. 두 번째 방법은 정렬된 용어목록을 화일로 저장하여 순차적으로 접근하여 검색하므로 서비스 속도가 빠르다. 하지만 데이터베이스의 변경된 사항을 동적으로 반영할 수 없다는 단점이 있다. 마지막 방법은 데이터베이스의 변경된 사황을 동적으로 반영할 수

있으며, 서비스 속도도 빠르다.

본 논문에서 구현한 서버는 마지막 방법을 사용하여 스캔 서비스를 구현하였다. 이런 경우 데이터베이스의 내용이 수시로 변경되더라도 변경된 데이터베이스 내용을 자동으로 용어 목록에 반영되고 서비스 속도도 빠르다.

(그림 8)은 스캔서비스가 구현 모듈을 나타낸 것이다. 각 데이터베이스마다 자체 용어목록을 가지고 있고, 서버는 클라이언트의 스캔 요구에 해당하는 데이터베이스의 용어목록에 접근하여 응답한다.

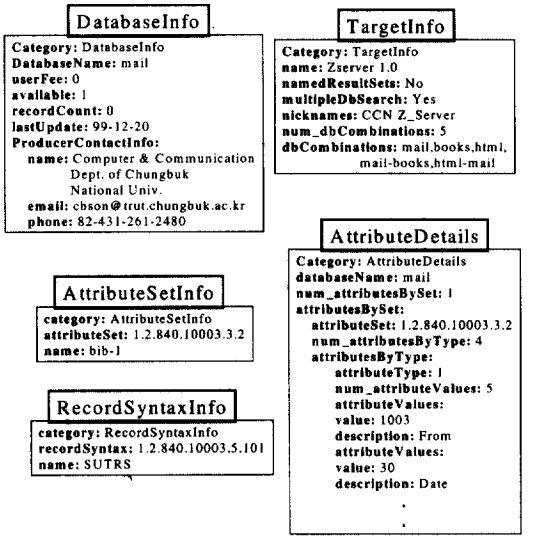


(그림 8) 스캔 서비스

4.2.5 설명기능

설명 기능은 클라이언트가 특정 서버에 대한 정보를 검색하게 해준다. 프로토콜에서는 서버에 대한 정보를 크게 16개의 범주로 분류하였다. 이 16개의 범주 외에도 필요에 따라 범주를 추가할 수 있도록 Category라는 지시자를 제공한다. 설명 기능에서 제공하는 서비스들을 간략하게 요약해 보면 서버에 관한 일반적인 정보, 사용 가능한 데이터베이스, 지원되는 속성 집합, 레코드 구문에 대한 정보 등을 제공한다. 이런 정보들은 IR-Explain-1이라는 이름의 데이터베이스에 저장된다. 그리고 이 데이터베이스에 대한 검색은 탐색기능과 검색기능을 사용한다. 본 논문에서는 다음의 TargetInfo, DatabaseInfo, RecordSyntaxInfo, AttributeSetInfo, AttributeDetails 그리고, CategoryList 이렇게 6개의 범주를 중심으로 데이터베이스를 구성하였다. 이상의 6개 범주는 프로토콜에 미리 정의된 것들이다.

프로토콜에는 미리 정의된 정보 범주에 의해 설명 데이터베이스를 검색할 때 클라이언트가 따라야 할 규칙들을 정해놓고 있다. 본 논문에서 구현한 서버에서 사용할 수 있는 규칙들을 <표 4>에서 보여주고 있으며, 이런 규칙들을 사용하여 구현한 서버에 질의할 경우 제공하는 설명 기능에 대한 결과들을 (그림 9)에서 제시하고 있다. 또한 클라이언트가 이 기능을 이용하여 질의할 경우에 Explain 레코드 구문을 지정해야만 원하는 결과를 얻을 수 있다.



(그림 9) 구현된 Explain Database의 범주별 레코드들

<표 4> 설명 데이터베이스 검색시 클라이언트가 따라야 할 규칙들

찾고자 하는 정보	애트리뷰트와 용어의 조합
서버에 관한 정보	ExplainCategory = "TargetInfo"
특정 데이터베이스에 대한 정보	ExplainCategory = "DatabaseInfo" and DatabaseName = "database name"
특정 레코드 구문	ExplainCategory = "RecordSyntaxInfo" and RecordSyntaxOID = "OID"
특정 속성 집합	ExplainCategory = "AttributeSetInfo" and AttributeSetOID = "OID"
특정 데이터베이스를 찾을 때 사용가능한 속성	ExplainCategory = "AttributeDetails" and DatabaseName = "database name"

4.3 구현 결과

본 절에서는 구현한 COBRA를 이용한 Z39.50 서버의 객체 구현에 대한 소스코드를 제시한다. 또한 서버의 우수성을 검증하기 위해 Z39.50 클라이언트의 상용 제품인 Znavigator 1.1H[6]와 BookWhere? 2000 Version 2.1[7]를 가지고 각각 테스트한 결과를 제시한다. (그림 10)은 객체 구현의 verity_search() 함수의 소스

코드이다. 소스코드에서 보면 검색엔진 Verity API가 사용된 함수는 veri_open()과 verity_search() 함수이다. Verity API를 사용하여 veri_open() 함수 소스코드는 (그림 11)에서 보여주고 있다. (그림 10)의 verity_search()함수는 Z39.50 서버의 제어 모듈에서 search_result = verity->verity_search(*base, Q)로 호출된다.

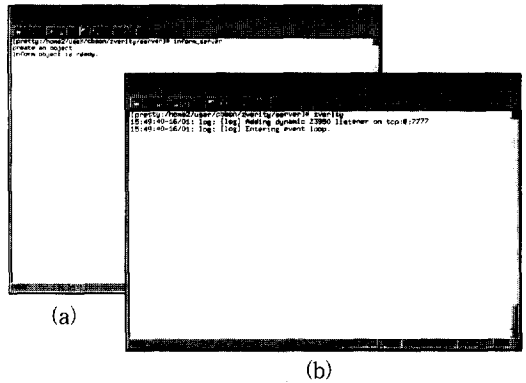
```
Verity::search_result* verity_search(const char *basename,
        const char* query){

          Verity::search_result *result = new search_result;
          veri_searchresult vs;
          char base_path[40] = "/home2/user/cbson/zverity/
          dbase/ mail";
          veri_open(path);
          vs = ver_search(query);
          result->count = vs.vs_hits;
          result->error_code = vs.vs_errcode;
          result->error_string = vs.vs_errstring;
          return result;
}
```

(그림 10) 객체 구현의 verity_search( ) 함수의 소스 코드

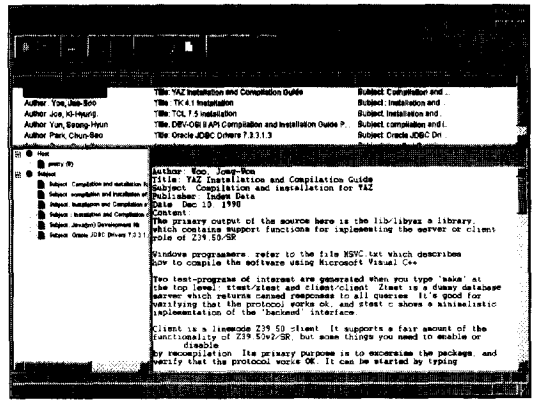
```
void veri_open(const char *ezcoll)
{

    VdkError error;
    VdkCollectionOpenArgRec collOpen;
    VdkStructInit(&collOpen);
    collOpen.path = (VdkCString)ezcoll;
    collOpen.serviceLevel = VdkServiceType_Search;
    error = VdkCollectionOpen(sessionHdl, &collectionHdl,
            &collOpen);
    if (error != VdkSuccess)
    {
        printf(vdkErrorMsg, "VdkCollectionOpen", error);
    }
}
```

(그림 11) Verity API을 이용한 veri_open() 함수 소스 코드

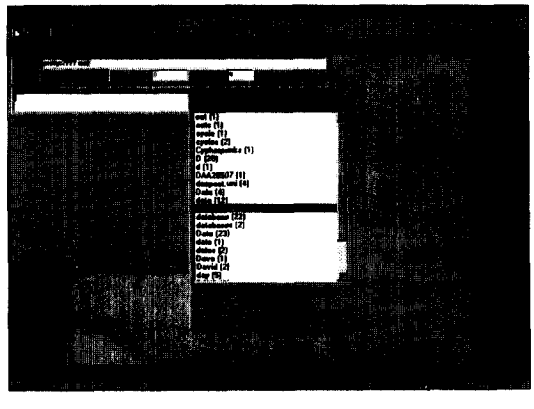
(그림 12-a)는 Verity API로 구현한 CORBA 서버 애플리케이션(객체 구현)을 수행시킨 모습을 보여주고 있고, (그림 12-b)는 Z39.50 서버가 실행되어 클라이언트의 요청을 기다리고 있는 상태를 나타내고 있다. (그림 13)은 상용 클라이언트인 Bookwhere 2000 ?을 가지고 Z39.50 서버에 있는 book 데이터베이스에 검색어 mail에 대해 검색을 수행한 결과를 제시한다. (그림 14)는 mail 데이터베이스에 검색어 database에 대한 스캔서비스의 결과를 보여 주고 있다.



(그림 12) 객체구현과 Z39.50 서버가 실행된 모습



(그림 13) BookWhere 2000?으로 검색된 결과



(그림 14) database에 대한 스캔서비스 결과

5. 결 론

본 논문에서는 Z39.50 프로토콜을 지원하는 Z39.50 서버의 데이터베이스 엔진모듈을 CORBA를 이용하여 통합 및 구현하였다. 이를 통하여 기존 Z39.50 클라이언트를 코드 수정없이 사용할 수 있으며, 이질적인 데이터베이스 엔진모듈을 쉽게 통합할 수 있어 시스템 확장에 많은 편리함을 제공하고 기존 데이터베이스 엔진 모듈 부분을 그대로 사용할 수 있어 자원의 재사용성을 높일 수 있다. 또한 CORBA 객체 구현을 분산시킴으로써 서버의 처리 부담을 줄일 뿐만 아니라 객체 구현에 대한 로케이션 투명성을 제공하고 데이터베이스 엔진 모듈의 변경이 투명하다. 구현한 Z39.50 서버를 테스트하기 위해 상용 Z39.50 클라이언트인 Book Where 2000?과 Znavigator를 가지고 테스트한 결과들을 제시하였다.

향후 연구 방향으로 구현된 Z39.50 서버는 Z39.50 프로토콜의 기능 중에서 가장 기본적인 서비스인 접속, 탐색, 전송, 종료서비스 외에 분할서비스, 스캔서비스, 설명기능을 지원하지만 표준에 정의되어 있는 모든 서비스들을 지원할 수 있도록 IDL 설계 및 통합 방법에 관한 것이다. 또한 현재 구현한 서버가 웹을 지원하도록 Z39.50/web 게이트웨이를 추가로 구현하는 것이다.

참 고 문 헌

[1] ANSI/NISO Z39.50-1995, ANSI Z39.50 : Information Retrieval Service and Protocol, 1995. http://lcewb.loc.lov/z3950/agency.

[2] 손충범, "정보검색 표준 프로토콜 Z39.50 서버 설계 및 구현", 석사학위논문, 1999.

[3] John A. Kunze, *Basic Z39.50 Server Concepts and Creation,* NIST Z39.50 Implementation papers, http://lcweb.loc.gov/z3950/agency/papers/nist.html.

[4] 박재현, 코아코바, 영한출판사, 서울, 1998.

[5] Infrise, Programmer's guide for VisiBroker for C++, manual, 1998.

[6] Znavigator software, CaseLibrary, http://www.sbu.ac.uk/litc/caselib/index.html.

[7] BookWhere 2000?, Sea Change Corp., http://www.bookwhere.com/bworder.htm.

[8] Denis Lynch, "Implementing Explain," *Distributed Search Conference*, http://lcweb.loc.gov/z3950/agency/papers/nist.html, 1996.

[9] Margaret St. Pierre, "Z39.50 for Full-Text Search and Retrieval," *Distributed Search Conference*, http://lcweb.loc.gov/z3950/agency/papers/nist.html, 1996.

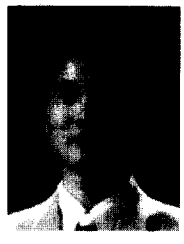
[10] Verity, *Search '97 Developer's kit getting started*, manual, 1996.

[11] Index Data, YAZ User's Guide and Reference, Index Data Corp, Document, 1997.

[12] Thomas J. Movbray, Ron Zahavi, The Essential CORBA : Systems Integration Using Distributed Object, John Wiloy & Sons, Inc., USA, 1995.

[13] OMG, The Common Object Request Broker : Architecture and specification Object management Group, Inc., Revision 2.0, July 1996.

[14] O2 Technology, O2 Engine API Reference Manual, manual, 1998.



손 충 범

e-mail : cbson@trut.chungbuk.ac.kr

1997년 충북대학교 정보통신공학과 (공학사)

1999년 충북대학교 정보통신공학과 (공학석사)

1999년~현재 충북대학교 정보통신공학과 박사과정

관심분야 : 데이터베이스 시스템, XML, 정보검색, 분산 객체 컴퓨팅

유 재 수

e-mail : yjs@cbucc.chungbuk.ac.kr

1989년 전북대학교 컴퓨터공학과 (학사)

1991년 한국과학기술원 전산학과 (공학석사)

1995년 한국과학기술원 전산학과 (공학박사)

1995년~1996년 목포대학교 전산통계학과 전임강사

1996년~현재 충북대학교 공과대학 전기전자공학부 조교수

관심분야 : 데이터베이스 시스템, 정보검색, 멀티미디어 데이터베이스, 분산 객체 컴퓨팅 등