CORBA를 이용한 OpenGIS기반 미들웨어 구현†

Implementation of OpenGIS-based middleware using CORBA

안경환(安耿煥)*, 조대수(曺大洙)*, 홍봉희(洪鳳憙)**

Kyoung-Hwan An, Dae-Soo Cho and Bong-Hee Hong

요 약 과거에는 공간데이타를 초기에 어떻게 구축할 것인가가 문제였으나 현재는 엄청난 양의 분산된 공간 데이타중 목적에 맞는 데이타에 대한 접근 및 효율적인 이용에 대해서 그 초점이 옮겨지고 있다. 이러한 요구를 만족하기 위해서는 분산 객체 기술과 공간 데이타 및 서비스에 대한 표준을 필요로한다. 이 논문에서는 OpenGIS에서 정의하는 표준 인터페이스를 분산 객체 기술인 CORBA를 이용하여 구현하는 방법에 대해서 다루고 있다. 클라이언트는 CORBA로 구현된 미들웨어의 OpenAPI를 표준 프로토콜인 IIOP(TCP/IP)를 통하여 접근할 수 있다. 클라이언트에게 Open API를 제공해 주는 CORBA 서버 객체는 기존 시스템을 포장하여 상호 운용을 보장해 준다.

ABSTRACT The focus is moving from the problem of building data to the problem of accessing and efficient utilization of data fitting to the purpose of a user. A distributed technology and a standard of geodata and geoprocessing is needed to satisfy these needs. This paper suggests an implementation method of a standard interface defined by OpenGIS with CORBA which is a distributed object technology. A client can invoke Open API of the middleware implemented with CORBA through IIOP. CORBA server objects which provide a client with Open API is the essence of an interoperability. CORBA server objects wrap the data formats and access methods of different GIS engines, and translate the data formats of different data servers to a standard data format.

키워드 : 상호운용, 개방형, CORBA, OpenGIS

1. 서 론

현재 공간 데이타를 이용하는 사용자와 조직의 수가 늘어남에 따라 공간 데이타에 대한 집적량이 급속도로 늘어가고 있다. 따라서 과거에는 공간 데이타를 초기에 어떻게 구축할 것인가에 대한 문제에서 현재에는 엄청난 양의 분산된 공간 데이타 중 목적에 맞는 데이타에 대한 접근 및 효율적인 이용에 관한 문제로 그 초점이 옮겨지고 있다. 그러나 기존에 구축된 시스템들은 나름대로의 데이타 포맷, 변환 방법, 데이타 처리 방법을 가지고 있으며, 소프트웨어가 사용되어지는 목적에 따라 그 복잡도가 점점 더해져 왔다. 이러한 폐쇄성과 복잡성으로 인해 기존에 구축된 시스템간의 공간 데이타 공유가 거의 불가능해졌다.

이러한 문제를 극복하기 위해서 기존에는 데이타 변환(transfer)에 의존해왔다. 대표적인 예로 SDTS (Spatial Data Transfer Standard)를 들 수 있는데, 이는 상이한 데이타 포맷을 가지는 시스템들간에 데이타를 주고 받기 위해 표준 데이타 포맷을 이용하는 방법이다. 그러나 이 방법의 문제점으로는 전체 데이타셋에 대한 일괄적인 처리 방법으로, 필요 없는 데이타까지 처리하는 비능률성과 데이타에 대한 중복 저장을 들 수 있다.

† 본 연구는 정통부의 대학기초연구지원사업의 연구비와 부산대학교 기성회재원 연구비의 지원으로 이루어 졌음.

* 준회원, 부산대학교 컴퓨터공학과

khan@hyowon.cc.pusan.ac.kr dsjo@hyowon.cc.pusan.ac.kr

** 종신회원, 부산대학교 컴퓨터공학과 교수

bhhong@hyowon.cc.pusan.ac.kr

현재에는 클라이언트/서버 구조와 함께 네트웍을 통한 데이타셋에 대한 직접적인 접근을 허락하고, 질의를 통해 필요한 데이타만을 가져오는 방법을 필요로 하고 있다. 이 방법을 위해서는 서로 다른 소프트웨어, 하드웨어 및 운영 체제들상의 시스템들에서 제공되는 서비스들을 이용할 수 있어야 하는데, 현재 분산 컴퓨팅 플랫폼(DCP: Distributed Computing Platforms)과 객체 기술은 이를 가능하게 해주고 있다. 또한 표준을 통해 여러 서버들에 대한 통일된 인터페이스를 클라이언트에게 제공해줄 필요가 있다.

상호 운용이란 위에서 말한 바와 같이 공간 데이타에 대한 표준 뿐만 아니라 서비스에 대한 표준 API를 정의함으로써 분산된 자원에 대한 공유를 가능하게 해주는 것을 말한다. 상호 운용을 지원하기 위해서는 각 시스템에서 공통적으로 이용할 수 있는 공간 데이타 및 서비스에 대한 표준이 필요한데 OGC(OpenGIS Consortium)의 OpenGIS(Open Geodata Interoperability Specification)가 이를 제공해 줄 수 있다[1][2][3]. OpenGIS에서는 모든 DCP 환경에 독립적인 추상 명세(Abstract Specification)와 함께 각 정보기술별 구현명세를 제시하고 있다. 현재에는 CORBA, OLE/COM, SQL의 세가지 구현 명세가 나와 있다. 본 논문에서는 이들 중 CORBA 구현 명세를 분석하고 그 구현 방법에 대해서 논의한다. 본 논문의 구성은 다음과 같다. 먼저 2장에서는 본 논문과 관련된 구현 사례에 대해 살펴보고, 3장에서는 CORBA 구현 명세의 표준 데이타 모델에 대해서 설명한다. 4장에서는 이를 구현하기 위한 전체적인 시스템의 구성에 대해서 그리고 5장에서는 표준 인터페이스를 이용하여 실제로 구현하는 방법에 대해서 기술한다. 마지막으로 6장에서 결론을 맺는다.

2. 관련 연구

기존에 다양한 데이터소스의 통합에 관한 연구는 많이 수행되어져 왔다. 그 중 대표적인 것은 multidatabase 접근 기법이다[7]. 이 기법은 데이타베이스들 위에 존재하면서 그것을 이용하는 사용자에게 하나의 데이타베이스가 존재하는 것처럼 보이게 한다. 이러한 multidatabase의 특징으로는 사용자가 질의를 하기위한 전역 스키마(global schema)를 유지한다는 것이다. 그러나 이 접근 방법의 문제점으로는 데이타베이스 기능을 가지지 못하는 데이타소스는 다루지 못한다는 것이다. 이러한 제한을 극복하기 위해 최근에는 분산 객체 기술을 이용하여 하부의 데이타소스를 캡슐화(encapsulation)하는 랩퍼(wrapper)에 관한 연구가 수행되고 있다[8][9]. 만약 데이타소스가 데이타베이스의 능력을 가지지 못할 때에는 랩퍼에서 이를 구현해주어 이 랩퍼를 이용하는 사용자에게 데이타소스에 대한 투명성을 제공해 줄 수 있다. 현재 대표적으로 이용되고 있는 분산 객체 기술로는 CORBA와 OLE/COM이 있다. 그렇지만 분산 객체 기술을 이용한 랩퍼의 연구에도 문제점이 존재한다. 첫번째로 이러한 랩퍼의 경우 기존의 속성 정보만을 다루는 일반적인 데이타베이스에 관한 연구이므로, GIS에 특정한 공간 데이타를 다루고 있지 않다. 두번째로는 클라이언트로 노출되어지는 인터페이스가 표준화되어 있지 않다는 것이다. 표준화된 인터페이스를 사용하지 않을 경우 클라이언트가 하나의 랩퍼에 종속되는 결과를 가져오게 되며 확장성이 결여되게 된다. 본 논문에서는 데이타베이스 기능을 가진 데이타소스뿐만 아니라 가지지 못한 데이타소스를 그 대상으로 하며, 분산 객체 기술인 CORBA를 이용하며, 클라이언트에게 노출되어 지는 랩퍼 즉 미들웨어 인터페이스로 OpenGIS의 표준 명세를 이용함으로써 기존 연구의 문제점들을 해결하고자 한다.

3. 표준 데이타 모델

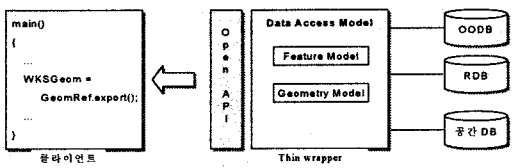


그림 1. Open API를 이용하는 클라이언트

본 논문에서는 그림 1과 같이 클라이언트로 하여금 DB 서버가 OODB, RDB, 공간 DB인가에 상관 없이 표준 데이타 접근 모델을 사용함으로써 상호 운용을 지켜주도록 하였다. 그림에서 보는 바와 같이 서로 다른 DB 서버를 포장(wrapping)하는 표준 데이타 모델은 OpenGIS의 피처 모델과 기하 모델로 이루어져 있다. 피처 모델은 DB 에 저장된 테이블의 한 행이나 객체들을 피처로 모델링하고 있으며, 기하 모델은 점, 선, 면과 같은 객체가 가지는 공간 속성을 모델링하고 있다. 여기서 피처 모델과 기하 모델은 OpenGIS 국제 표준에서 구현 명세로 이미 제시된 내용이다. OpenGIS의 CORBA 구현 명세에서는 CORBA IDL(Interface Definition Language)로 OpenGIS의 구성 요소들을 표현하고 있으며, 이러한 IDL은 피처 모델과 기하 모델과 같은 표준

데이타 모델을 클라이언트에게 노출(expose)시켜주고 있다.

3.1 피처 모델

피처 모델은 GIS DB에 대해 피처와 피처 컬렉션들을 생성, 접근, 질의하기 위한 구성 요소들로 이루어져 있으며, 그림 2와 같은 인터페이스들로 정의되어 있다. 피처 모델은 크게 클라이언트에서 GIS DB에 대한 접근을 가능하게 해주는 ContainerFeatureCollection과 질의를 가능하게 해주는 QueryableContainerFeatureCollection, 피처 객체의 속성 및 기하 정보에 대한 접근을 가능하게 해주는 Feature 인터페이스와 피처 객체의 생성을 위한 FeatureFactory, FeatureType 인터페이스들로 구성되어 있다. 위의 여러 인터페이스 중 Feature 인터페이스 IDL의 예를 들면 아래와 같다. 클라이언트에서는 피처에 속한 속성을 읽어 오기 위해서 get_propery() 인터페이스를 이용하고, 기하 데이타를 읽어 오기 위해서는 get_geometry()를 이용한다.

```
interface Feature
{
    ...
    //feature type
    readonly attribute  FeatureType     feature_type;
    //geometry
    Geometry            get_geometry(in NVPairSeq
                         geometry_context) raises (InvalidParams);
    // properties
    boolean             property_exists(in Istring name)
                         raises(InvalidProperty);
    any                 get_property(in Istring name)
                         raises (PropertyNotSet, InvalidProperty);
...
    NVPairSeq           get_property_sequence(in unsigned long n);
    FeaturePropertySetIterator      get_property_Iterator();
    void destroy();
};
```

현재 다른 구현명세와 CORBA 구현 명세를 비교해보면 이 피처 모델에서 차이가 난다는 것을 알 수 있다. OLE/COM을 위한 구현 명세의 경우에는 피처 모델에 해당하는 명세가 존재하지 않는다. 대신 OLE DB라는 기존에 이미 정의되어 있는 데이타 접근 모델을 이용하고, 단지 공간데이타 관련 부분만 OLE DB에 추가로 정의해 놓고 있다. 그러나 CORBA 구현 명세에서는 피처 모델내에 QueryEvaluator와 같은 인터페이스를 새로이

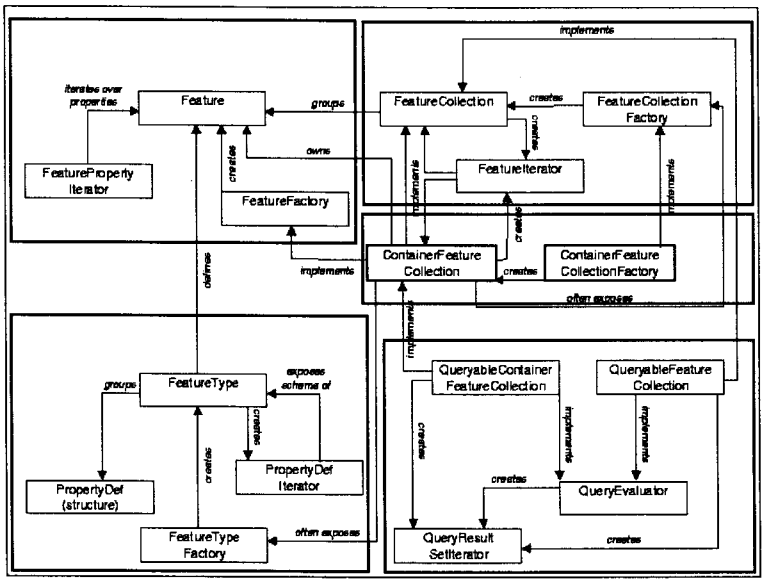


그림 2. 피처 모델의 인터페이스 관계[3]

정의하고 이를 통해서 데이타에 접근하는 방법을 채택하고 있다. 즉 OpenGIS의 구현 명세에서는 데이타에 접근하기 위한 방법은 최대한 기존에 이미 정의되어진 정보기술의 방법을 사용한다고 볼 수 있다.

3.2 기하 모델

상호 운용이 가능한 GIS를 위해서는 모호하지 않는 기하 객체 타입과 연산을 클라이언트로 노출할 필요가 있다. OpenGIS에서 기하는 그들의 차원에 따라 점(Point), 선(Curve), 면(Surface)등으로 분류되며 이런 기하들이 모여 복합(composite) 기하 또는 기하 컬렉션을 이룬다. 또한 모든 기하들은 그들의 좌표 기하(coordinate geometry)를 WKS(Well Known Structure)로 노출할 수 있으며, 의미는 SRS(Spatial Reference System)에 의해서 제공되어 진다. 이러한 기하 관련 인터페이스들중에 Geometry 인터페이스는 아래와 같다. 이 인터페이스를 살펴보면 두 개의 기하간의 공간 연산자들도 포함된다는 것을 알 수 있다.

```
interface Geometry
{
    ...
    readonly attribute short dimension;              // dimension of
                                                        the geometry
    readonly attribute Envelope range_envelope; // minBoundBox
                                                        in abstract spec
    readonly attribute SpatialReferenceSystem
                                        spatial_reference_system;
    // geometric characteristics
    boolean is_closed();
...
    // constructive operators
    Geometry copy();
    ...
    // WKS operators
    WKSGeometry export();
...
    // relational operators
    boolean touches (in Geometry other);
    boolean contains (in Geometry other);
    boolean within (in Geometry other);
...
    // metric operators
    double distance (in Geometry other);
    // set operators
    Geometry intersection (in Geometry other);
    ...
    void destroy();
};
```

4. 시스템 구조

기존의 시스템을 포장하여 상호운용을 지원하기 위한 전체적인 기본 구조는 그림 3(a)와 같다. 이 구조에서 가장 핵심은 계층2의 OpenAPI를 클라이언트에게 제공하여 주는 데이타제공자에 있다. 데이타제공자는 하부의 각 GIS 서버와는 고유의 폐쇄된 인터페이스를 통해 질의하고, 그 결과를 얻게 되며, 클라이언트로부터는 개방된 표준 인터페이스를 통해서 질의 및 데이타를 주고 받는다.

본 논문에서의 데이타제공자는 CORBA 구현명세의 표준 인터페이스를 구현하는 CORBA 서버 객체이다. 그림 3(b)에서 보는 바와 같이 CORBA 서버 객체가 데이타제공자를 이루고, 각 CORBA 서버 객체는 GIS 서버인 Gothic, ORACLE등의 API를 이용하여 표준 인터페이스를 이용하여 접근하는 클라이언트의 요청을 처리해 주고 있다. 클라이언트가 접근할 수 있는 대표적인 CORBA 서버 객체의 인터페이스로는 ContainerFeature Collection과 QueryableContainerFeatureCollection이 있는데 이들 인터페이스 구현 객체의 내부 구조를 살펴보면 그림 4와 같다.

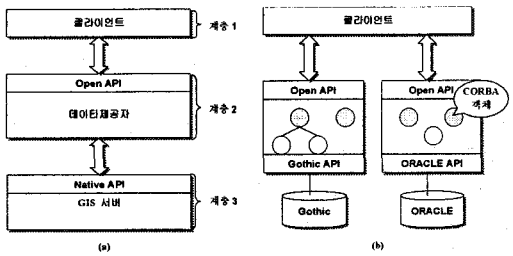


그림 3. 시스템 구조

그림4를 보면 크게 Open API를 구현하는 부분과 데이타 변환 모듈 그리고 DB API 모듈이 존재한다. DB API의 경우 GIS 서버 벤더들에 의해서 제공되어지는 라이브러리등을 의미하고, 데이타 변환 모듈의 경우 각 GIS 서버 고유의 데이타 포맷을 WKS의 형태로 또는 WKS를 각 GIS 서버 고유의 포맷으로 변환하는 역할을 담당한다. 인터페이스 구현 모듈은 표준 인터페이스를 구현하는 모듈로서 데이타 변환 모듈에서 변환되어진 데이타를 클라이언트의 요구에 따라 제공하는 역할을 담당한다. 결국 데이타제공자에 해당하는 CORBA 서버 객체들은 하부 GIS 서버의 종류나 구조에 상관없이 일관된 인터페이스를 제공하는 역할을 하고 있다.

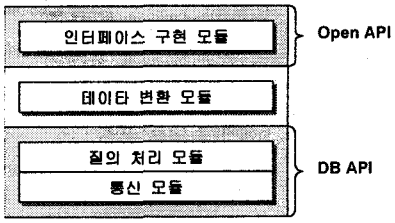


그림 4. QueryableContainerFeatureCollection의 내부 모듈

본 논문에서는 구현을 위해서 CORBA로는 Inprise사의 visibroker를 이용하고 있으며, GIS 서버로는 Laser Scan사의 Gothic을 이용하고 있다. 고딕상에서 공간 SQL의 처리를 위해서는 GSQL 서버[4][10]를 이용하고 있다. 클라이언트와 서버 객체의 구현을 위한 언어로는 자바를 이용하고 있다.

5. 구현 방법

5.1 용어 정리

본 논문에서 쓰여질 용어 및 표기 방법을 정리해 보면 다음과 같다.

① *OpenGIS 구현 명세의 IDL*
- 클라이언트측에서 이용할 수 있는 서비스 함수에 대한 인터페이스들의 정의로, 본 논문에서는 이를 영문으로 표기하고 있다.

② *피처 객체*
- 실제 GIS DB에 저장된 데이타를 의미한다. GIS DB가 OODBMS일 경우 DB에 저장된 객체를, RDBMS일 경우 테이블의 한 행을 의미한다.

③ *피처 인터페이스*
- GIS DB에 저장된 피처 객체의 속성 또는 기하정보를 읽어 올 수 있도록 해주는 서비스 함수들의 집합으로 IDL에서의 Feature를 의미한다.

④ *피처 서버 객체*
- 피처 인터페이스를 구현한 CORBA 서버 객체를 의미 한다.

⑤ *WKS(Well Known Structure)*
- 클라이언트와 서버간의 피처 객체의 전송시 쓰이는 표준 데이타 포맷을 말한다.

⑥ *GSQL*
- [4]에서 정의된 공간 질의어 문법으로 SQL에 기하와 관련된 문법이 추가되어 있다.

5.2 공간 자원 발견

네트워크상에 존재하는 다양한 데이타 소스들 중에 클라이언트가 원하는 자원에 접근하기 위해서는 특별한 방법을 필요로 한다. 본 논문에서 기반으로 하는 CORBA 구현 명세에서는 이러한 공간 자원을 발견하기 위한 메커니즘으로 CORBA의 객체 서비스인 이름 서비스(naming service)와 거래 서비스(trading service)를 이용한다고 되어 있다. 이름 서비스의 경우 ORB상에 존재하는 다른 CORBA 구현 객체를 찾는 방법으로 객체의 이름으로 참조(reference)를 얻을 수 있도록 해주는 서비스이다. 거래 서비스의 경우 새로운 서비스를 제공하는 CORBA 구현 객체가 먼저 거래자(trader)에게 객체 참조와 제공되는 서비스에 대한 설명을 등록하게 된다. 그런 뒤 클라이언트가 거래자에게 제공되는 서비스들의 목록을 요청하거나 원하는 서비스를 요청하면 그와 가장 유사한 서비스를 클라이언트에게 제공해주는 서비스이다. 현재 ORB를 구현한 상용 제품들의 경우 아직까지 CORBA 객체 서비스들을 많이 지원해 주고 있지는 않다. 이름 서비스의 경우 대부분의 회사들이 이를 지원해 주고 있지만 거래 서비스의 경우는 대부분 지원하지 못하고 있는 실정이다. 따라서 본 논문에서는 이름 서비스를 이용하여 공간 자원을 발견하는 방법을 제시한다.

5.2.1 구성 방법

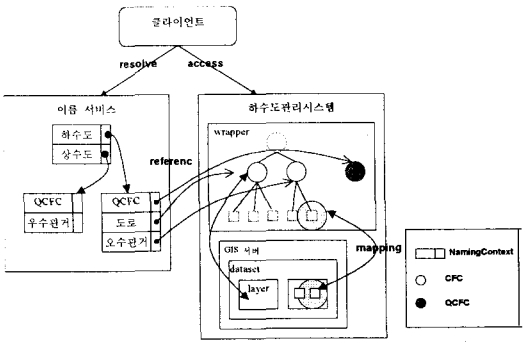


그림 5. 이름서비스를 이용하여 객체에 바인딩하는 방법

그림 5는 클라이언트가 이름서비스를 이용하여 원하는 데이타를 제공해줄 수 있는 서버 객체에 바인딩하는 방법을 보여주고 있다. 먼저 이러한 서비스가 가능하기 위해서는 데이타베이스 관리자(Database Administrator)가 상호 운용을 원하는 데이타제공자 서버 객체들을 이름 서비스에 이름과 함께 등록을 하여야 한다. 현재 CORBA 구현 명세에서는 이름 서비스에 등록을 하기 위한 이름에 대하여 구체적인 명세를 제시하고 있지 않다. 이름 서비스에서는 일반적으로 NamingContext를 이용하여 계층적인 이름을 지원하는 데 본 논문에서는 등록을 위한 이름을 아래와 같은 규칙을 이용하여 정하였다.

- 1단계 NamingContext :

첫번째 단계에는 상호운용하기를 원하는 시스템들의 데이타셋 이름을 등록하도록 하였다.

- 2단계 NamingContext :
  두번째 단계에서는 각 시스템의 데이타셋 내의 레이어 이름 또는 질의를 위한 CORBA 서버 객체 이름인 QueryableContainerFeatureCollection을 넣도록 하였다. 만약 이 단계에서 레이어명을 이름으로 등록할 때는 해당하는 ContainerFeatureCollection 서버 객체의 참조를, 질의를 위해서는 QueryableContainerFeatureCollection을 구현한 서버 객체에 대한 참조를 등록하도록 하였다.

예를 들어, 클라이언트에서 상수도 관리 시스템과 하수도 관리 시스템이라는 두가지의 서버로부터 공간 데이타를 가져와야 한다고 가정했을 때, 이름 서비스에 등록되는 최상위의 NamingContext는 상수도, 하수도라는 두가지 데이타셋에 대한 이름이 될 것이다. 그런 다음 두번째 단계에서는 각 시스템별로 가지고 있는 모든 레이어의 이름과 각 레이어를 포장하고 있는 ContainerFeatureCollection 또는 QueryableContainerFeatureCollection에 대한 참조를 가지고 있는 NamingContext를 등록하게 된다.

5.2.2 이용 방법

클라이언트에서는 아래와 같은 시나리오에 의해서 원하는 공간 데이타 및 서비스를 제공하는 서버객체에 바인딩할 수 있다.

① 먼저 클라이언트가 이름 서비스를 이용하기 전에 데이타베이스 관리자는 각 시스템의 데이타셋 이름, 레이어명, 각 레이어를 포장하는 CORBA 서버 객체의 참조를 이름서비스에 등록한다.

② 클라이언트가 만약 각 GIS 서버의 데이타셋명과 레이어명을 안다면, {{데이타셋명},{레이어명}}의 조합을 이용하여 resolve() 인터페이스 호출을 통해 객체 참조를 얻어 올 수 있으며, 최소한 {데이타셋명}을 이용하여, 해당 NamingContext를 얻은 다음 CORBA 이름 서비스의 인터페이스인 BindingList 와 BindingListIterator를 이용하여 원하는 객체 참조를 얻을 수 있다.

③ 클라이언트는 얻어진 객체 참조를 이용하여 질의를 수행하거나, 피처객체를 얻어온다.

5.3 피처 객체 얻어오기

5.3.1 CORBA 서버 객체와 GIS DB의 매핑

CORBA 서버 객체는 기존의 GIS 서버를 포장하여 클라이언트에게 표준 인터페이스를 지원하는 가상의 GIS 서버인 것처럼 보이게 해준다. 이러한 CORBA 서버 객체들은 결국 하부의 GIS 서버내에 저장되어 있는 객체와 매핑되어 있어야 사용자의 질의에 응답 할 수 있게 되는데 문제가 되는 것은 그 단위를 어떻게 하는가 이다. 왜냐하면 단위에 따라 CORBA 서버 객체에서 해야하는 일이 달라지기 때문이다. 본 논문에서는 그림 6과 같은 방법을 택하고 있다.

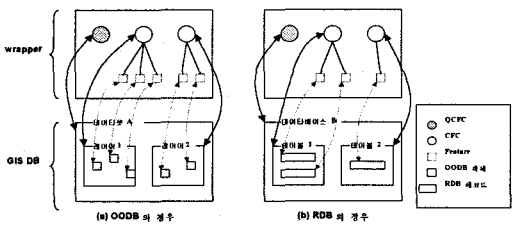


그림 6. CORBA 서버 객체의 매핑

먼저 QueryableContainerFeatureCollection의 경우에는 GIS 서버의 데이타셋 또는 데이타베이스에 매핑된다. 즉 서버 객체가 사용자의 질의 요청을 받았을 때, 매핑되어 있는 데이타셋을 대상으로 질의를 GIS DB로 보내고 그 결과를 받게 된다. ContainerFeatureCollection의 경우 하나의 레이어 또는 테이블에 매핑이 되는데 사용자가 이 서버객체에 바인딩 했을 때 접근하는 단위는 하나의 레이어가 된다. 또한 사용자가 OODB의 객체 또는 RDB의 레코드에 접근하고자 할 때, 이에 매핑되는 피처 서버 객체의 경우는 필요에 따라 동적으로 생성되어 사용자에게 데이타를 제공해준다. 피처 서버 객체의 경우 동적으로 생성될 때, OODB의 경우 OID 값을, RDB 일 경우 기본 키를 멤버 변수값으로 가지도록해서 클라이언트가 객체나 레코드의 속성값을 얻고자 할 경우 이 멤버 변수값을 이용하여 GIS DB로부터 얻어오도록 하였다.

5.3.2 일반적인 피처 객체 얻기

CORBA 서버 객체중에서 클라이언트에게 노출되는 인터페이스는 ContainerFeatureCollection이 있다. ContainerFeatureCollection은 일반적인 피처 브라우저와 같은 곳에 쓰이며, 저장된 피처 객체 전체에 대한 순차적인 접근이나, 피처 객체를 GIS DB내에 저장하고자 할 때 등의 용도로 쓰일 수 있다.

ContainerFeatureCollection에 필요한 FeatureCollection의 구현 방법으로는 두가지가 있을 수 있다. 현재 컬렉션에 대한 기본적인 구조로 자바의 Vector 클래스를 이

용하고 있는데, 이 Vector에 들어가는 요소가 OID(하나의 객체를 나타내는 유일한 값, 관계형 데이타베이스의 경우 기본키)인지 Feature 서버 객체인지에 따라 두가지로 나뉘어 진다.

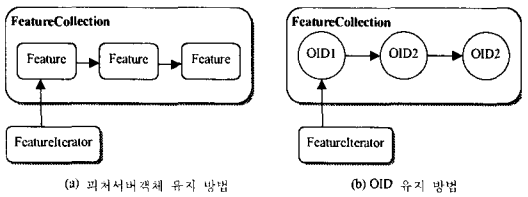


그림 7. FeatureCollection의 구현 방법

첫번째로 그림 7 (a)의 경우 클라이언트로부터 피처 서버 객체에 대한 요구가 왔을 때 내부에 유지하고 있던 피처 서버 객체에 대한 참조를 넘겨 주는 방법이다. 그렇지만 이 방법은 FeatureCollection 내에 미리 모든 피처 서버 객체를 생성하여 유지해야 하는데 만약 클라이언트에서 모든 피처 서버 객체를 필요로 하지 않을 경우 생성 및 유지 비용이 너무 크다는 단점이 있다. 그림 7 (b)는 OID만을 내부적으로 유지하고 있다가 클라이언트로부터 요청이 왔을 때 동적으로 피처 서버 객체를 생성하여 그 참조를 넘겨 주는 방법으로 유지 비용을 줄일 수 있다. 본 논문에서는 그림 8에서와 같이 OID테이블이라는 OID를 유지하는 두번째 구현방법을 사용하고 있다.

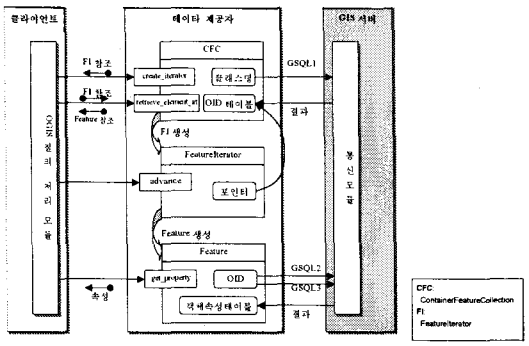


그림 8. 피처 객체를 얻기 위한 메커니즘

아래에서는 ContainerFeatureCollection을 이용하여 피처 객체에 접근하여 사용하는 방법을 두가지로 나누어 설명한다. 첫째는 클라이언트에서 피처 서버 객체를 얻어 오는 것이고, 둘째는 얻어온 피처 서버 객체를 이용해서 속성 정보 및 기하정보를 얻어 오는 것이다.

첫번째로 피처 서버객체를 얻어 오기 위한 과정은 다음과 같다.

① ContainerFeatureCollection에 바인딩

먼저 클라이언트는 피처 서버 객체를 얻어오기 위해서 서버로부터 노출된 ContainerFeatureCollection에 바인딩해야 한다. 이 때 서버 객체에 바인딩하는 방법은 앞에서 설명한 이름서비스를 이용한다.

② FeatureIterator 생성

클라이언트는 create_iterator() 메시지를 Container FeatureCollection 서버 객체에 보내면 그 서버 객체는 레이어명을 이용하여 GIS 서버에 질의하여 그 레이어에 속한 OID들을 읽어오고, FeatureIterator 서버 객체를 생성한 뒤, 그 참조를 클라이언트에게 넘겨 주게 된다. 다음은 ContainerFeatureCollection 서버 객체가 GIS 서버로 보내는 GSQL의 예제로 그림에서 GSQL1에 해당한다.

```
SELECT 건물.OID
FROM 건물;
```

③ 피처 서버 객체 참조 얻기

클라이언트는 retrieve_element_at()을 FeatureIterator 참조와 함께 서버에 보내면, ContainerFeatureCollection은 GIS DB에 저장되어 있는 피처 객체에 매핑되는 피처 서버 객체에 대한 참조를 넘겨주게 된다. 피처 서버 객체를 생성할 때 GIS DB에 저장된 피처 객체 중에 어떤 것과 매핑되는지를 알 수 있도록 하기 위해서 내부적으로 가지고 있던 OID 테이블내의 한 OID값을 피처 서버 객체내의 내부 멤버 변수에 설정을 해주게 된다.

두번째로 속성 및 기하정보를 얻기 위한 과정은 다음과 같다. 먼저 클라이언트에서 피처 객체의 속성을 얻고자 할 경우, 피처 서버 객체에 get_property() 메시지를 보내면 되는데, 이 때 내부에 유지하고 있던 OID를 이용 계층3의 GIS 서버에 질의를 보내 속성 데이타를 읽어오도록 하였다. 다음은 속성 정보를 읽기 위해 GIS 서버로 전송하는 GSQL의 예제로 그림에서 GSQL2에 해당한다.

```
SELECT 건물.소유주
FROM 건물
WHERE 건물.OID = "111.222.333";
```

사용자가 공간 연산 및 기하 정보가 필요하다고 할 경우 피처 인터페이스의 get_geometry()를 이용하여 Geometry 인터페이스를 구현한 서버 객체에 대한 참조를 얻어 올 것이다. 물론 이 때에도 피처 서버 객체 내에 유지하고 있던 피처 객체에 대한 OID를 Geometry 서버 객체 생성시에 멤버 변수값으로 설정 해준다. 그렇

게 함으로써 실연 목적에 의해서 기하 데이타를 클라이언트로 가져올 필요가 있을 때 Geometry 인터페이스에 정의되어 있는 export() 또는 export_WKBGeometry()를 사용해서 WKS(Well Known Structure) 또는 WKB (Well Known Binary Representation)의 형태로 데이타를 가져올 수 있다. 이 때도 Geometry 서버 객체 내의 OID를 이용해서 GIS 서버로 질의를 보내 기하 정보를 가져온 뒤 이를 WKS로 변환하여 결과를 돌려 주는 과정을 거치게 된다. 아래 예제는 기하 데이타를 가져오기 위해 GIS 서버로 전송하는 GSQL3의 예이다.

```
SELECT 건물.GEOMETRY
FROM 건물
WHERE 건물.OID = "111.222.333";
```

5.3.3 질의를 통해 얻기

클라이언트는 QueryableContainerFeatureCollection을 이용하여, 속성 및 공간 질의를 할 수 있는데, 일반적인 클라이언트의 경우 주로 이 인터페이스를 이용하여 원하는 데이타를 가져올 수 있다.

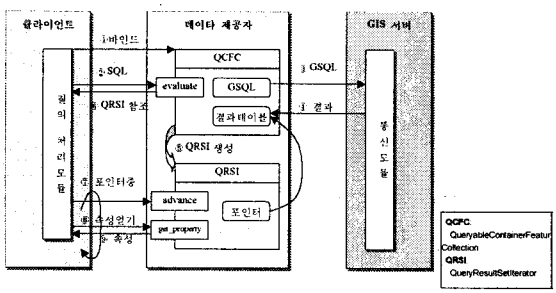


그림 9. 질의를 통해 객체를 얻기 위한 메커니즘

질의를 통해 피처 객체에 접근하기 위한 과정은 다음과 같다.

① 클라이언트는 먼저 QueryableContainerFeature Collection 서버 객체에 바인딩 해야 한다.

② 서버 객체내에 정의된 evaluate() 메소드를 이용하여 질의문을 전송한다.

③ QueryableContainerFeatureCollection은 클라이언트로부터 받은 SQL을 필요에따라 변환하여 GIS 서버의 인터페이스를 이용하여 전송한다. 다음은 GIS 서버로부터 데이타를 얻어오기 위한 GSQL의 예제이다.

```
SELECT 건물.GEOMETRY, 건물.소유주, 건물.준공일자
FROM 건물, 상수도관
WHERE 건물.소유주 ="안경환" AND CROSSES(건물,
                                    상수도관);
```

④ GIS 서버에서는 질의문을 수행한 결과를 테이블의 형태로 넘겨준다.

⑤ CORBA 서버 객체는 테이블 데이타에 대한 접근 방법을 가지고 있는 QueryResultSetIterator 서버 객체를 생성한다.

⑥ QueryResultSetIterator 참조를 클라이언트로 넘겨준다.

⑦ 데이타를 얻어오기 위해서 클라이언트는 Query ResultSetIterator내의 결과 테이블에 대한 내부 포인터를 증가 시켜주는 advance()를 호출한다.

⑧ 클라이언트는 get_property_by_name()등의 속성을 얻기위한 인터페이스 메소드를 호출한다.

⑨ QueryResultSetIterator는 QueryableContainerFeature Collection내의 테이블 데이타에 접근하여 클라이언트로 전달하여 준다.

⑩ 다른 속성을 얻기 위해서는 ⑦부터 반복한다.

5.4 공간 연산자의 실행

OpenGIS CORBA 구현명세에는 공간 연산자를 실행하기 위한 인터페이스가 2가지 존재한다. 하나는 Geometry 인터페이스 내에 존재하는 공간 연산자들이고 다른 하나는 QueryEvaluator 인터페이스에 존재하는 query_by_properties()에 의해서이다.

Geometry 인터페이스내에 존재하는 공간 연산자의 경우는 두개의 기하간에 연산을 수행하여 참/거짓을 돌려주는 연산이다. 이 공간 연산자들의 경우 간단한 두개의 기하에 대한 비교연산이므로 미들웨어인 CORBA 서버객체내에서 구현하게된다. 그렇게 함으로써 특정 GIS 서버에 따라 중복 구현의 필요성이 없어진다. query_by_properties() 인터페이스 경우는 공간 연산의 대상이 전체 데이타셋에 대해서이다. 이 경우 성능 향상을 위해서는 하부 GIS 엔진의 공간 연산 처리 기능을 이용하는 것이 바람직할 것이다. 그러므로 이 인터페이스를 이용하여 공간 연산을 실행할 때에 미들웨어에서는 각 GIS 엔진에 맞는 공간 질의로 변환하여 GIS 서버로 보내어 질의를 처리한 뒤 그 결과를 사용자에게 되돌려주는 방법을 취하게 된다.

6. 결 론

본 논문에서는 이질적인 GIS 서버간에 데이타 및 서비스의 공유를 위해 OGC의 OpenGIS를 이용하였다. 현재 OpenGIS에는 CORBA, OLE/COM, SQL의 세가지 구현 명세가 존재하는데 본 논문에서는 소프트웨어 및 하드웨어, 운영체제등에 독립적인 CORBA를 이용하여 표준 인터페이스를 구현하는 방법을 다룬다. CORBA 구현 명세에서 클라이언트에게 노출되어지는 표준 인터페이스는 IDL로 정의되어 있으며, 이를 구현하는 CORBA 서버 객체는 기존 시스템들을 포장하는 가상의 GIS 서버 역할을 하게 된다. 이들 CORBA 서버 객체들 중 클라이언트에게 처음 노출되어지는 인터페이스로는 ContainerFeatureCollection과 QueryableContainer FeatureCollection이 있는데 전자는 일반적인 피처 브라우저나 저장된 피처 객체에 대한 순차적인 접근 또는 피처 객체를 GIS DB내에 저장하고자 할 때 쓰이며, 후자는 질의를 통하여 데이타를 얻고자 할 때 주로 쓰인다. 본 논문에서는 이러한 서버 객체에 바인딩하는 방법으로 이름 서비스를 이용하는 방법을 제시하고 있으며, 각 인터페이스를 구현하는 방법 및 문제점을 제시하고 있다.

향후 연구로는 계속되는 OpenGIS 표준화 작업의 진행에 따라 구현 방법에 대한 연구 및 갱신이 필요할 것이다. 또한 서로 다른 구현 명세들에 대한 비교 분석을 통해서 다른 DCP 환경의 구현 방법에 대해서도 연구가 필요하다. 마지막으로 실제로 여러 GIS 서버에 대한 실험을 통하여 상호 운용에 대한 입증이 필요하다.

참 고 문 헌

[1] Open GIS Consortium, Inc., The OpenGIS Guide, 1998.

[2] Open GIS Consortium, Inc., The OpenGIS Abstract Specification Model, Version 3, 1998.

[3] Open GIS Consortium, Inc., OpenGIS Simple Features Specification for CORBA, Revision 1.0, 1998.

[4] 조대수, 최진오, 류우석, 안경환, 홍봉희, 객체 지향 GIS DB를 위한 GSQL의 설계 및 구현, 데이타베이스 연구회지 13권 4호, pp51-63.

[5] 안경환, 조대수, 홍봉희, CORBA를 이용한 클라이언트/서버 GIS의 설계 및 구현, 한국정보과학회 98 가을학술 발표 논문집 vol 25, no 2, pp185-187, 1998.

[6] 안경환, 조대수, 홍봉희, 상호 운용을 지원하는 CORBA 기반 WWW GIS의 설계, 한국 개방형 GIS 연구회 학술회의 논문집 1권 1호, pp.199-209, 1998.

[7] Ling Liu, L. Yan, and M.T. Ozsu., Interoperability in Large-Scale Distributed Information Delivery Systems, In Advances in Workflow Systems and Interoperability, 1998.

[8] Asuman Dogac, Cevdet Dengi, M. Tamer Ozsu, Building Interoperable Databases on Distributed Object Management Platforms, Communication of ACM, 1996.

[9] Yooshin Lee and Ling Liu, Calton Pu, Toward Interoperable Heterogeneous Information Systems: An Experiment Using the DIOM Approach, the Proceesings of the 12th ACM Symposium on Applied Computing (ACM SAC'97), 1997.

[10] 조대수, 홍봉희, 고딕 기반 인터넷 GIS의 구현, 98 동계 데이타베이스 학술대회 발표논문집 14(1), pp33-40,1998.

[11] Zhong-Ren Peng, An Assessment of the Development of Internet GIS, Proc. of the 1997 ESRI User Conference, 1997.

[12] David J.DeWitt, Navin Kabra, Jun Luo, Jignesh M. Patel, and Jie-Bing Yu, Client-Server Paradise, Proceedings of the 20th VLDB Conference, 1994.

[13] Ted G. Lewis, Where Is Client/Server Software Headed?, COMPUTER vol 28, 1995.

[14] Jan Kleindienst, Frantisek Plasil, Petr Tuma, Lessons Learned from Implementing the CORBA Persistent Object Service, OOPSLA '96, 1996.

[15] Jurgen Sellentin, Bernhard Mitschang, Data-Intensive Intra- & Internet Applications - Experiences Using Java and CORBA in the World Wide Web -, 4th International Conference Data Engineering, 1998.

[16] 송창빈, 김기홍, 전주용, 김주관, 권용식, 차상균, SDBC: 복수의 공간 OODBMS 접속을 위한 개방형 미들웨어, 한국정보과학회 '97 가을 학술 발표 논문집(I) 24권 2호, pp63-66, 1997.

[17] 유정연, 이상, 이강찬, 이규철, CORBA와 DBMS 연동 방법, 정보과학회 '98 봄 학술 발표논문집(B), pp140-142, 1998.

[18] 신혜균, 장지훈, 김정선, 최중민, 우훈식, CORBA

기반 데이타베이스 브로커의 설계, 정보과학회 '98 봄 학술 발표 논문집(B), pp158-160, 1998.

[19] R. Orfali, D.Harkey, Client Server Programming with JAVA and CORBA, John Wiley and Sons, 1997.

[20] R. Orfali, D. Harkey, J. Edwards, The Essential Distributed Objects Survival Guide, John Wiley and Sons, 1997.

[21] MICHAEL F. WORBOYS, "GIS: A Computing Perspective", Taylor & Francis, 1995.

[22] VisiBroker for Java Reference Manual Version 3.2, visigenic.

[23] VisiBroker for Java Programmer's Guide Version 3.2, visigenic.

[24] http://www.opengis.org.

[25] http://www.omg.org.

안경환

1997년 부산대학교 컴퓨터공학과 졸업 (공학사)

1999년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사)

1999년~현재 부산대학교 대학원 컴퓨터 공학과, 박사과정

관심분야 : GIS, 분산객체기술, 개방형지리정보시스템

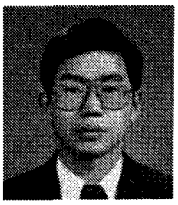
조대수

1995년 부산대학교 컴퓨터공학과 졸업 (공학사)

1997년 부산대학교 대학원 컴퓨터공학과 졸업(공학석사)

1997년~현재 부산대학교 대학원 컴퓨터 공학과, 박사과정

관심분야 : 객체지향 공간 질의어, 인터넷 GIS, 공간 데이타 모델링



홍봉희

1982년 서울대학교 전자계산기공학과 졸업(공학사)

1984년 서울대학교 대학원 전자계산 기공학과 졸업(공학석사)

1988년 서울대학교 대학원 전자계산 기공학과 졸업(공학박사)

1987년~현재 부산대학교 공과대학 컴퓨터공학과 교수

관심분야 : 공간 데이타베이스, GIS표준화, 병렬 GIS, 개방형지리정보시스템