2000년 한국정보처리학회 추계 학술발표논문집 제7권 제2호

공간 정보유통을 위한 CORBA 서버

오병우*, 이종훈*
*한국전자통신연구원 GIS 연구팀
e-mail: bwoh@etri.re.kr

A CORBA Server for Spatial Clearinghouse

Byoung-Woo Oh* and Jong-Hun Lee*
*GIS Research Team, ETRI (Electronics and Telecommunications Research Institute)

요약

지리 정보 시스템의 발전으로 공간 데이터의 수요가 늘어남에 따라 공간 데이터의 공급자들이 늘어나고 있다. 일반적으로 공간 데이터의 공급자는 수요자의 요구에 맞는 형식을 사용하여 공간 데이터를 생산한다. 그래서, 생산된 공간 데이터는 다양한 플랫폼에서 다양한 형식으로 저장된다. 최근들어 분산환경에서 다양한 플랫폼에 저장된 공간 데이터를 효율적으로 접근하기 위하여 공간 정보유통의 필요성이 증대되었고 OpenGIS Consortium(OGC)은 공간 데이터의 상호호환성을 위해 ISO와 협력하며 국제 표준화를 추진중이다.

본 논문에서는 공간 정보유통 중 공간 데이터 접근을 위해 OGC의 Simple Features Specification for CORBA 표준을 기반으로 ESRI사의 공간 데이터 저장 형식인 Shape 파일을 위한 CORBA 데이터 프로바이더를 구현한다. CORBA는 언어와 플랫폼에 독립적으로서 분산환경에서의 상호운용성을 제공하고, 특히 Java와 결합하면 웹 브라우저를 통해 어떤 플랫폼에서든지 사용 가능하다. 그러므로, Shape 파일에 대한 CORBA 서버를 개발함으로써 플랫폼에 종속되지 않는 공간 데이터의 접근을 제공할 수 있다. 본 논문에서는 설계시 고려사항 및 구현 방법에 대해 살펴보고 OGC 표준을 구현하면서 발견된 문제점 및 성능 향상을 위한 방법에 대해서도 언급하여 다른 공간 데이터 형식을 위한 CORBA 서버를 개발할 때 참조할 수 있도록 한다.

1. 서론

최근 GIS에 대한 연구가 각 분야에서 연구되고 있으며, 특히 컴포넌트 개념을 사용하여 재사용성을 증대하고, 개방형 GIS에 관한 연구를 통해 상호운용성을 제공하는 연구가 활발히 진행중이다. GIS의 상호운용성을 위한 공간 정보유통은 분산환경에서 네트워크를 통해 원하는 공간 데이터를 검색하고 검색된 데이터를 접근하는 기술로 정의할 수 있다 [1]. 효율적인 공간 정보유통을 위해서는 위치 및 플랫폼에 상관없이 공간 데이터에 접근하기 위한 기술이 가장 중요하다.

본 논문에서는 CORBA를 사용하여 원격지의 공간 데이터를 접근할 수 있도록 해주는 공간 정보유통을 위한 데이터 프로바이더를 개발한다. 공간 정보유통을 위한 CORBA 서버는 ESRI사의 Shape 파일 형식으로 저장된 공간 데이터를 Open GIS Consortium (OGC) 표준에 맞는 인터페이스를 통해 제공하는 역할을 담당한다.

본 논문의 구성은 다음과 같다. 2장에서는 OGC에 대해 간략히 살펴보고, 3장에서는 설계시 고려한 사항에 대해 언급한다. 4장에서는 CORBA의 원격 객체 접근을 위한 늦은 처리 속도를 감안한 성능 향상 구현 기법을 제안한다. 5장에서는 실제 구현을 통해 발견된 OGC 표준의 미비점 및 해결 방안에 대해 언급한다. 6장에서는 결론 및 앞으로의 연구 방향을 제시한다.

2. OpenGIS Consortium (OGC)

OGC(http://www.opengis.org)는 국제적인 200여 기관이 설립한 단체로서 GIS에 대한 실질 (de

facto) 표준을 제정하기 위한 노력을 기울이고 있다. OGC에서는 추상 표준으로부터 구현을 위한 인터페이스 표준까지 다양한 표준을 개발하고 있으며 ISO와의 친밀한 협조 체제를 이룩하여 국제 표준으로 발전시키고 있다. 본 논문에서는 OGC의 구현 사양 중에서 CORBA를 위한 단순 형상 (Simple Features) 사양을 구현한다 [2].

OGC의 용어 중에서 Feature는 관계형 데이터베이스의 튜플(tuple)로 간주하면 되고, Feature Collection은 테이블 또는 뷰, Feature Iterator는 커서(cursor), FeatureType은 스키마(schema), Property는 필드(field)에 해당한다. 특히, Feature Collection은 GIS에서 흔히 레이어(layer)로 부르는 개념으로서 실제 Feature를 소유하지는 않고 참조 포인터만 가지고 있다. 반면, Container Feature Collection은 실제로 Feature를 소유하는 테이블로 간주된다.

3. 공간 정보유통을 위한 CORBA 서버의 설계

OpenGIS Simple Features Specification for CORBA 표준은 크게 Feature와 Geometry(공간 참조 포함)의 2부분으로 나뉘어 진다. 본 논문에서는 그림 1과 같은 Feature 모델을 구현한다.

Feature 모델에서 사용자는 요구에 맞는 다양한 인터페이스를 통해 공간 데이터에 접근할 수 있다. 그런데, Shape 파일은 DBMS에 의해서 관리되지 않고 파일 시스템에 직접 저장되어 질의 처리가 제공되지 않으므로 QuaryableContainerFeatureCollection 인터페이스를 구현하는 것이 어렵고 구현 사양에서 ContainerFeatureCollection을 가장 많이 사용하게 될 것이라고 언급하고 있어 본 논문에서는 사용자가 ContainerFeatureCollection을 통해 접근하는 것을 가정하고 일련의 작업을 설계하였다.

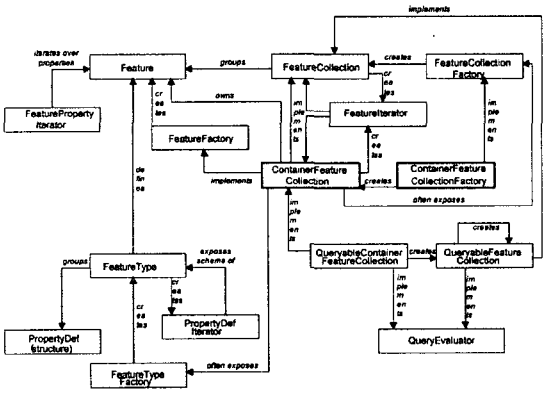

그림 1. OGC의 Feature 모델

사용자는 그림 1에서 보는 바와 같이 ContainerFeatureCollectionFactory 인터페이스를 통해 ContainerFeatureCollection을 생성하여 작업하도록 설계하였다. OGC의 Feature 모델에서는 Feature Collection도 하나의 Feature로 간주하고 Feature로부터 상속받으므로 공간 데이터의 처리는 속도 향상을 위해 ContainerFeatureCollection 인터페이스의 get_wkbgeometry_by_name() 메소드를 한번만 호출하여 OGIS::WKSCollectionType으로 넘어온 Well-Known-Binary (WKB) 구조를 처리하도록 설계하였다. 비공간 데이터는 FeatureIterator를 생성하여 각각의 Feature들을 처리하도록 설계하였다.

4. 공간 정보유통을 위한 CORBA 서버의 구현

본 논문에서는 MS-Windows 98 환경에서 Borland C++ Builder 5.0과 Visibroker for C++ 4.0을 사용하여 구현하였다.

공간 정보유통을 위한 CORBA 서버를 위하여 FeatureTypeFactory, FeatureType, PropertyDefIterator, ContainerFeatureCollectionFactory, ContainerFeatureCollection, FeatureIterator, Feature 인터페이스를 순서대로 구현하였다.

4.1 FeatureTypeFactoryServer

ContainerFeatureCollection을 생성하기 위해서 사용되는 ContainerFeatureCollectionFactory의 create() 메소드를 호출하기 위해서는 FeatureType이 필요하므로 FeatureType을 생성하기 위하여 FeatureTypeFactory 인터페이스를 구현하였다.

FeatureTypeFactory 인터페이스는 create() 메소드 하나만을 포함한다. create() 메소드는 데이터베이스에 연결(connect)하는 역할을 담당하며 FeatureTypeImpl 클래스의 인스턴스를 생성하고 Shape 파일 처리 객체, 스키마 정보를 저장하는 Property 정의 시퀀스 등의 멤버 변수들을 초기화하는 작업을 수행한다.

4.2 FeatureTypeServer

FeatureType 인터페이스는 주로 스키마 정보를 알기 위한 Property 관련 메소드를 제공하므로 이를 구현하였다. 다만, Shape 파일은 상속의 개념이 존재하지 않으므로 부모 Feature Type을 얻기 위한 메소드 등의 상속 관련 인터페이스는 길이가 0인 Feature Type 시퀀스를 반환하는 함수로 구현하였다.

4.3 PropertyDefIteratorServer

FeatureType 인터페이스 중에서 get_property_def_iterator() 함수를 사용하여 반환된 결과를 사용할 수 있도록 하기 위하여 구현하였다.

그러나, 매번 remote의 CORBA 서버를 호출하여 PropertyDefIterator를 사용하기보다는 FeatureType의 get_property_def_sequence() 함수를 사용하여 Property 정의 시퀀스를 local로 가져온 후에 처리하는 것이 성능향상에 도움이 될 것으로 생각된다.

4.4 ContainerFeatureCollectionFactoryServer

Container Feature Collection을 생성하기 위한 ContainerFeatureCollectionFactory 인터페이스 중에서는 create() 메소드만을 구현하였다. 나머지 메소드는 Collection, FeatureData, Sequence로부터 Container Feature Collection을 생성하기 위한 것으로 Shape 파일과는 관련이 적어 구현하지 않았다.

ContainerFeatureCollectionFactory에서는 ContainerFeatureCollectionImpl 클래스의 멤버 변수인 feature_type을 세팅하기 위해서 FeatureTypeImpl 객체의 포인터가 필요하므로 매개변수로 넘어온 FeatureType_ptr인 _collection_type으로부터 다음과 같은 과정을 통해 포인터를 얻어낸다. 즉, 서버측에서 FeatureTypeImpl(Feature Type의 Servant)의 객체를 생성하고 클라이언트에는 FeatureType으로 넘겨졌던 객체를 다시 매개변수로 받아서 서버측의 FeatureTypeImpl로 변환하기 위해서는 다음과 같은 과정이 필요하다.

```
PortableServer::ServantBase_ptr servant;
servant = _collection_type->_preinvoke("");
feature_type = (FeatureTypeImpl *)
                OGIS::FeatureType_ops::_downcast(servant);
```

4.5 ContainerFeatureCollectionServer

ContainerFeatureCollection 인터페이스는 FeatureCollection과 Feature로부터 상속받으므로 FeatureIterator 생성, Feature Element에 대한 add, delete 등의 작업을 수행한다.

특히, Feature 인터페이스로부터 상속받은 get_wkbgeometry_by_name() 메소드를 성능향상을 위해 구현하여, Iterator를 통해 Feature 한 개씩 접근하며 WKB를 얻어오지 않고 ContainerFeatureCollection에 해당하는 모든 Feature들의 WKB를 CORBA 호출 한번으로 모두 local로 읽어올 수 있도록 하였다. 즉, GIS에서 가장 빈번하고 중요한 작업인 공간 데이터를 읽어오기 위한 과정을 한번의 CORBA 호출로 처리하므로 매우 효율적인 공간 데이터 처리가 가능하다.

4.6 FeatureIteratorServer

ContainerFeatureCollection에 속한 Feature들에 한 개씩 또는 지정된 수만큼 접근하기 위한 Iterator를 구현하였다. next()와 next_n() 메소드에서는 FeatureImpl 객체를 생성하여 클라이언트의 요구를 처리할 수 있도록 하였다. get_<type>_by_name() 함수(예를 들면, get_string_by_name())에서는 현재 가리키고 있는 Feature의 Property 값을 FeatureImpl 객체를 생성하지 않고 직접 얻을 수 있도록 구현하였다.

4.7 FeatureServer

Feature의 Property 값을 얻기 위한 get_<type>_by_name() 함수들을 구현하였다. 그러나, 성능을 위해서는 Feature 객체를 생성하고 값을 얻는 것보다 FeatureIterator 인터페이스에 있는 함수들을 사용하는 것이 좋을 것으로 생각된다.

4.8 테스트 클라이언트 개발

공간 정보유통을 위한 CORBA 서버의 테스트를 위하여 그림 2와 같은 클라이언트를 개발하였다. 클라이언트는 접근하고자 하는 Shape 파일 이름을 입력받아 CORBA 서버를 호출하는 일련의 작업을 수행하여 공간 데이터를 출력하고 그 시간을 표시해 준다. 시간은 공간 데이터를 얻기 위해 걸린 총시간과 이를 객체의 수로 나눠서 한 개의 객체를 처리하는데 걸린 시간을 출력한다. 그리고, 사용자가 비공간 데이터를 원할 경우 이를 테이블 형태로 출력한다.

그림 3은 공간 정보유통을 위한 CORBA 서버를 사용하여 공간 데이터를 출력하는 예제이다.

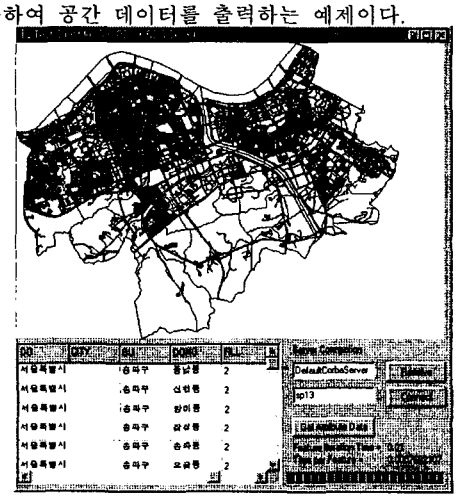

그림 2. 테스트 클라이언트 실행 화면

```
featureType = featureTypeFactory->create(
                                    feature_name,
                                    p_def_null,
                                    parent);
p_def = featureType->get_property_def_sequence(0, 0);
containerFeatureCollection =
                containerFeatureCollectionFactory->create(
                                    featureType,
                                    nv_null);
wkb = containerFeatureCollection->
                get_wkbgeometry_by_name(geom_property);
p = wkb->get_buffer();
p++; // skip byte_order
// skip wkbType (7)
p += sizeof(unsigned long);
// num_wkbGeometries
n = *(unsigned long *)p;
p += sizeof(unsigned long);
for (i = 0L; i < n; i++) {
    next = display_WKB(p, Image1);
    p += next;
}
OGIS::OctetSeq::_release(wkb);
```

그림 3. 클라이언트 예제

5. 문제점 및 해결방안

5.1 CORBA 호출 시간

CORBA는 분산 환경에서 remote의 객체를 local에 있는 것처럼 쉽게 사용할 수 있도록 해주는 강력한 기능을 지원하지만 이를 처리하기 위해 많은 시간을 소모한다. 단순히 remote의 메소드를 한번 부르기 위하여 CORBA가 처리하는 데 상당한 시간이 필요하며 CORBA 객체 생성에도 많은 시간이 소모된다. Feature Iterator의 next() 메소드를 사용해 처리하려면 "Feature 개수 × (CORBA 호출 처리시간 + Feature 객체 생성시간 + 실제 작업)" 만큼의 시간이 걸리므로 Feature 개수가 늘어날수록 처리 시간이 크게 증대된다. next_n() 메소드를 사용하면 어느 정도 줄일 수 있을 것으로 기대 했지만 Feature 객체를 생성하는데 드는 시간은 줄일 수 없으므로 만족할만한 성능 향상은 얻을 수 없었다. advance()를 사용한 후에 get_wkbgeometry_by_name()을 사용하면 Feature 객체 생성에 소모되는 시간을 줄일 수 있지만 CORBA 호출 자체에 걸리는 시간을 줄일 수는 없다.

성능 향상을 위해 가장 효율적인 방법은 Container Feature Collection이 Feature로부터 상속받고 Feature Collection 자체를 하나의 Feature로 볼 수 있으므로 ContainerFeatureCollection 인터페이스의 get_wkbgeometry_by_name()을 사용하여 모든 wkb를 한번에 받아와서 local로 처리하는 방법이다. 이 방법의 문제점은 메모리 소모가 많다는 것이다.

5.2 메타데이터 부족

어떤 Feature Collection들을 지원하는지에 대한 리스트를 얻을 수 없다. 즉, 어떤 Shape 파일들에 접근 가능한지 알 수 없다. 또한, 공간 데이터를 출력하기 위해 필요한 Feature Collection의 map extent(즉, 레이어에 대한 MBR)를 제공하지 않아 처리가 힘들다. 이러한 문제들은 공간 정보유통의 카탈로그 검색 서비스와 결합하여 해결할 수 있다. 공간 정보유통으로 검색된 메타데이터를 통해 어떤 공간 데이터가 제공되는지 알 수 있을 것이고, 메타데이터의 필드에 Interoperable Object Reference (IOR)을 추가하여 CORBA 서버에 대한 접근을 제공하며, map extent도 메타데이터에 추가하여 공간 데이터 처리에 도움을 줄 수 있을 것이다.

6. 결론 및 향후연구방향

본 논문에서는 OGC의 OpenGIS Simple Features Specification for CORBA 표준 중에서 Feature 모델 부분을 구현하며 표준에서 추가로 필요한 부분을 살펴보고 성능 향상을 위한 구현상의 방법에 대해 언급하였다.

향후연구방향으로는 OLE/COM 데이터 프로바이더[3]를 위한 CORBA 서버(OLE/COM Data Provider Wrapper)를 개발하여 플랫폼-독립적으로 사용할 수 있도록 발전시키는 것과 공간 정보유통의 카탈로그 서비스와의 연계가 있다.

Acknowledgement

본 논문은 정보통신부의 개방형 GIS 컴포넌트 S/W 과제 및 NGIS 표준화 - 개방형 GIS 컴포넌트 과제에 의해 지원 받았습니다.

실험에 사용할 수 있도록 다양한 양질의 공간 데이터를 제공해 주신 (주)한국통신정보기술에 감사드립니다.

참고문헌

[1] OpenGIS Consortium, OpenGIS - Catalog Interface Implementation Specification, Revision 1.0, 2000.

[2] OpenGIS Consortium, OpenGIS Simple Features Specification for CORBA, Revision 1.1 (DRAFT), 1999 (http://www.opengis.org/techno/interop/dsto/sfcorba-11-99-054.pdf).

[3] OpenGIS Consortium, OpenGIS Simple Feature Specification for OLE/COM, Revision 1.1, 1999.