JAVA를 이용한 CORBA 구현명세 기반의 분산객체GIS 구현방안

(A Development of the Distributed Object GIS based on CORBA Implementation Specification with JAVA)

손진락*, 정준원**, 진희채**

(Son JinRak*, Jung JunWon**, Jin Heui-Chae**)

초 록

지리정보 및 이와 관련된 프로세스의 표준화를 위해서 구성된 OGC에서는 개방형 GIS를 구현하기 위한 방법으로 CORBA, COM, 그리고 SQL의 세 가지를 제시하고 있다.

본 논문에서는 세 가지 구현방안 중 CORBA 구현명세를 바탕으로 하는 분산객체GIS에 대한 구현모델을 제시하고, 제시된 구현모델에 대하여 JAVA와 VisiBroker를 이용하여 서버와 클라이언트 측면에서 구현하는 방안을 제시하고자 한다.

키 워 드

분산객체GIS, JAVA, CORBA, VisiBroker, 구현방안

1. 서 론

OGC(Open GIS Consortium)에서 개방형 GIS의 구축을 위하여 추상명세(Abstract Spec.)와 구현명세(Implementation Spec.)의 두 가지 명세(Specification)를 개발하였다.

추상명세는 GIS를 구축하기 위한 개념적인 모델을 제시하는 것이 목적이고, 구현명세는 추상명세를 바탕으로 하여 실제 시스템을 개발할 수 있도록 지원해주는 것이 목적이다.

추상명세는 97년의 Version 2.0에서 98년 Version 3.0이 개정되면서 Topic 9 Accuracy가 Quality로 개편되었고, Topic 14가 추가되었고, 99년에 Topic 15, Topic 16이 추가되고 Version 4.0으로 개정되었다.[OGA99]

구현명세는 현재 CORBA, COM, SQL의 세 가지 방법으로 구현할 수 있도록 되어 있으며, 이중 CORBA Revision 1.0은 현재 Feature Model Architecture, Feature Interface 등 6개의 부분을 수정하여 Revision 1.1로 개정하는 작업을 진행 중에 있으며, COM과 SQL 버전은 99년에 Revision 1.1로 개정된 상태이다.[OGP99-041]

본 논문은 CORBA 구현명세 Revision 1.0을 기준으로 하여 JavaSoft의 JDK version 1.2.1과 Inprise의 VisiBroker Version 3.4를 이용하여 분산객체GIS를 구축하기 위한 구축모델과 클라이언트와 서버의 구현방안을 논하고자 한다.

JavaSoft의 JDK 1.2.1은 JAVA 2라도 한다. JAVA는 90년도에 그린프로젝트를 시작으로 95년도에 처음으로 공개되었으며, 96년도에 JDK 1.0.2를 발표하였다. 이후로 UNICODE 체계의

* 한국전산원 국가정보화센터
** 한국전산원 정보화평가분석단

수용, RMI, JFC, JDBC, Servlet, CORBA와의 연동, JavaBeans 등의 다양한 기능이 추가되었으며, 98년 12월에 JDK 1.2가 발표되었으며, 또한 98년에 ISO에서 JAVA기술을 국제표준으로 수용하기도 하였다.

이러한 JAVA는 다른 언어에 비해 구현이 간단하고, 완전한 객체지향적인 설계가 가능하고, 멀티쓰레드 기능 등의 고성능을 갖고 있으며, 인터프리트 언어로 실행코드(바이트코드) 수준에서의 플랫폼 호환성과 분산 네트워크 환경에서 프로그램을 개발할 수 있는 강점을 가지고 있다[정99].

Inprise사의 VisiBroker는 OMG(Object Management Group)의 CORBA 표준을 따르는 ORB제품으로 현재 C++와 JAVA 두 가지 언어로 Version 3.4가 발표되어 있다.[VBJ34]

CORBA는 OMG에서 개발된 단체표준으로 분산환경에서의 객체지향 시스템 개발을 위한 명세로 91년 CORBA 1.1을 시작으로 94년 CORBA 2.0이 제정되었으며, 현재 Version 2.2까지 발간되었으며, CORBA 3.0을 개발 중에 있다. CORBA 3.0에서는 분산컴포넌트의 지원, 새로운 메시징 지원, 타 분산환경과의 통합 등이 포함될 예정이다[박98]. VisiBroker는 현재 CORBA 2.0을 지원하고 있다[VBJ34]

본 논문의 구성은 다음과 같다. 2장에서는 OGC의 OpenGIS 추상명세 및 구현명세에 대해서 알아보고, 3장에서는 CORBA구현명세를 이용하여 분산객체GIS에 대한 구현모델을 제시하고, 이를 VisiBroker와 JAVA를 이용하여 실제적으로 시스템을 구현하는 방안을 제시한다. 4장에서는 구현방안에 대한 전체적인 논의와 발전방향, 그리고 결론을 맺는다.

2. OGC의 OpenGIS 추상명세 및 구현명세

OGC는 GIS S/W개발업체, DBMS개발업체 등 관련 산업체에서 지리정보처리에 있어서의 상호운용성을 확보하기 위하여 구성된 표준화 단체로, 지리정보처리를 위한 기술을 명세한 추상명세와 이를 각각의 플랫폼에서 구현하기 위한 구현명세를 개발하였다[OGA99].

2.1 OpenGIS 추상명세

추상명세는 구현명세의 개발에 필요한 개념적인 모델을 제공하기 위하여 Syntropy[1] 객체분석과 설계방법론에 따라 도출된 두 개의 모델로 이루어져 있다[OGA99].

첫 번째 모델은 기본모델(Essential Model)로, 실제세계에 대한 소프트웨어 또는 시스템 디자인의 개념적인 연결을 설정하는 것이고, 두 번째 모델은 추상명세에서 자세히 기술된 추상모델로, 구현방법, 구현시스템 등에 중립적인 방법으로 궁극의 소프트웨어시스템을 정의함으로써 실세계의 현상에 대한 소프트웨어와 시스템 디자인, 구조, 검색, 필터, 사례와 편집 등을 지원한다.

현재 OGC의 추상명세는 Version 4.0으로 99년 3월에 최근 개정되었다. 97년의 Version 2.0에서 98년 Version 3.0이 개정되면서 Topic 9 Accuracy가 Quality로 개편되고, Topic 10이 Topic 5와 관계뿐만 아니라 Topic 1,2,3,8,11과 연관되게 하였으며, Topic 14 Semantics and Information Communities가 추가되었고, 99년 Version 4.0이 되면서 Topic 15, Topic 16이 추가되었으며, 기존의 OMT방법론의 표기법에서 ISO 국제표준인 UML을 이용하여 표기하고 있다. 이외에 각 Topic내에서 부분적인 개정이 있었다.[OGA99]

OGC의 추상명세는 현재 17권(Topic)으로 구성되어 있으며, 그 구조 및 연관관계는 다음과 같다.

---

1) Syntropy은 영국의 Object Designers Limited에서 개발된 차세대 객체지향분석 및 설계방법론이다.

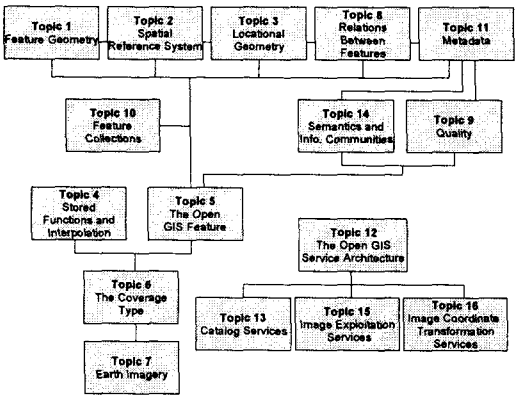

그림 1. 추상명세 Topic 연관도

2.2 OpenGIS 구현명세

OGC의 구현명세는 현재 SQL, COM, 그리고 CORBA의 세 가지 방법으로 구현할 수 있도록 되어 있으며, 98년 3월에 Revision 1.0이 확정된 후, COM과 SQL버전은 99년도에 Revision 1.1로 개정된 상태이고, CORBA 버전은 현재 Feature Model Architecture, Feature Interface, FeatureIteratior Interface 등 6개 항목에 대하여 개정을 추진하고 있다. [OGP99-041]

□ COM 구현명세

COM 구현명세는 Microsoft의 OLE/COM을 기반으로 하여 ODBC(Open DataBase Connectivity), DAO, OLE DB, ADO등의 기술을 이용하여 구현할 수 있도록 정의하고 있다.

그림 2에서 데이터제공자(Data Provider)는 실제 GIS 데이터를 가지며 OLE DB에서 정의된 인터페이스를 통해 일관된 방법으로 데이터에 접근할 수 있도록 하며, 서비스제공자 또는 소비자의 요구에 따라 WKB(Well-Known Binary)의 형태로 Geometry 값을 내놓게 된다.

서비스제공자(Service Provider)는 공간질의, 버퍼영역서비스, Geocoding서비스 등과 같은 지형공간 서비스를 제공한다. 소비자(Customer)는 직접적으로 데이터제공자를 이용할 수도 있고, 간접적으로 서비스제공자가 제공하는 각종 서비스를 이용하여 사용자가 필요로 하는 기능을 구현한다.

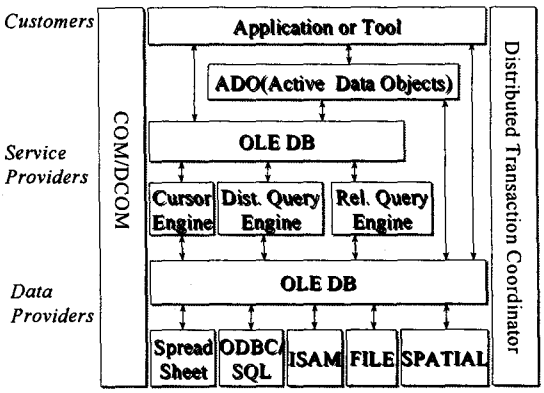

그림 2. OLE/COM환경에서의 Data Access Architecture

□ SQL 구현명세

SQL 구현명세는 ODBC API를 통하여 추상명세에서 정의된 지형공간정보에 대한 저장, 추출, 질의 등이 가능하도록 하기 위한 SQL표준 Schema를 정의하는 것이 목적이다. 공간데이터와 비공간데이터는 관계형DBMS의 Table에서 각 columns에 저장된다. SQL92로 정의된 Feature Table Schema는 그림 3과 같다.

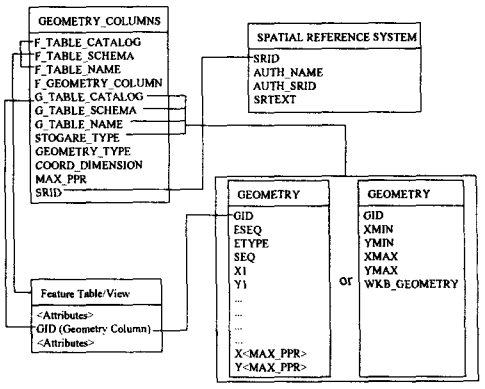

그림 3. SQL92 기반의 Feature Table에 대한 Schema

Feature table/view 는 OpenGIS feature 클래스에 해당된다. 각 feature view는 view에서의 row의 형태로 몇 개의 features를 포함한다. geometry는 좌표값을 갖거나 OpenGIS에서 WKB로 정의된 이진값을 갖는다.

□ CORBA 구현명세

CORBA 구현명세는 OMG의 CORBA를 기반으로 하여 분산객체GIS를 구축할 수 있도록 인터페이스를 정의한 명세이다.

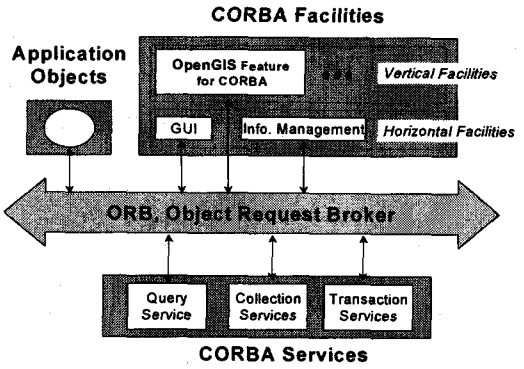

그림 4. OpenGIS Feature를 포함한 CORBA구조

CORBA 구현명세에서는 CORBA IDL (Interface Definition Language)로 Feature, Geometry, 그리고 SRS(Spatial Reference System)에 대한 표준인터페이스를 정의하고 있다. 이 명세는 구현에 있어 기존의 RDBMS등에 저장된 지형공간정보에 대하여 객체 Wrapping 등과 같은 기법을 사용하여 변환할 수 있는 등 광범위한 융통성을 제공하며, 프로그래밍언어, 운영체제 등에 독립적인 시스템을 구현할 수 있다.

3. CORBA를 이용한 분산객체GIS 구축 방안

3.1 분산객체 GIS 구현모델

CORBA를 이용한 분산객체 GIS를 구축하기 위해서는 여러 가지의 구현모델이 개발될 수 있다. 본 논문에서는 소프트웨어 개발의 효율성 등을 향상시키기 위하여 클라이언트의 경우 Package와 같은 형태로 개발하고, 서버의 경우 지형정보접근을 위한 부분을 분리하여 구현함으로써 데이터베이스에 독립적인 시스템을 구현할 수 있도록 다음과 같은 모델을 제시한다.

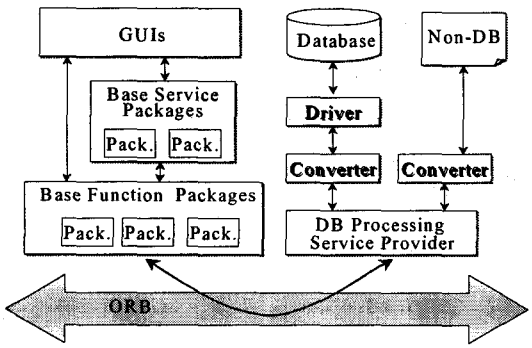

그림 5. 분산객체 GIS 구현모델

서버 측에서 DB Service Processing Provider는 Feature 데이터와 질의 데이터에 관련된 인터페이스를 구현하여 서비스를 제공하는 제공자이다. Converter는 기존의 RDBMS 등에 저장된 지형공간정보를 WKS의 형태로 바꾸어주는 역할을 한다.

클라이언트 측에서 Feature와 geometry에 대한 공간질의, 버퍼영역서비스, geocoding 서비스, 네트워크분석서비스를 수행하는 부분이 Base Function Package가 되고, 이들에 대한 공통적인 서비스를 제공하는 것이 Base Service Package가 된다.

즉, 클라이언트 프로그램에서 Basic Function Package를 이용하여 서버객체 쪽으로 Feature와 geometry에 기반을 둔 객체 서비스를 요구하게 된고, 서버측에서는 DB Service Processing Provider에서 클라이언트에서 요구된 Feature와 Geometry에 기반을 둔 객체 서비스를 수행하고 그 결과를 Basic Function Package에 되돌려 주

게 된다.

3.2 CORBA 프로그래밍

CORBA를 기반으로 하는 분산객체GIS를 구축하기 위해서는 OGC의 추상명세에서 정의된 Feature, Geometry, SRS 등을 IDL로 정의하는 것이 필요하다.(이것이 CORBA 구현명세이다.)

표 1. IDL의 구조

```
module <식별자>
{
   <형 선언>
   <상수 선언>
   <예외 선언>

   interface <식별자> [ : <상속> ]
   {
     <형 선언>
     <상수 선언>
     <속성 선언>
     <예외 선언>

     [<반환형>] <식별자> (<인수>)
     [raises (<예외>)][컨텍스트] ;
     ......
     [<반환형>] <식별자> (<인수>)
     [raises (<예외>)][컨텍스트] ;
   }

   interface <식별자> [ : <상속>]
   .......
}
```

Module은 여러 클래스들을 정의하기 위한 공간을 제공한다. Interface는 클라이언트가 호출하는 객체의 메쏘드(Method)를 정의한다. 메쏘드는 클라이언트가 호출할 수 있는 서비스를 나타낸다. 각 인수들은 클라이언트에서 서버로 전달(in), 서버에서 클라이언트로의 전달(out), 또는 두 가지 모두(inout) 가능하다. <반환형>은 반환되는 값의 형식을 나타내며, raises문은 메쏘드를 수행하면서 발생될 수 있는 예외를 옵션으로 정의할 수 있게 한다. [컨텍스트] 부분은 옵션으로 클라이언트의 컨텍스트 정보에 관련된 속성들을 포함한다. 이것은 클라이언트가 지역 환경에 관련된 정보를 서버로 전달하게 한다.

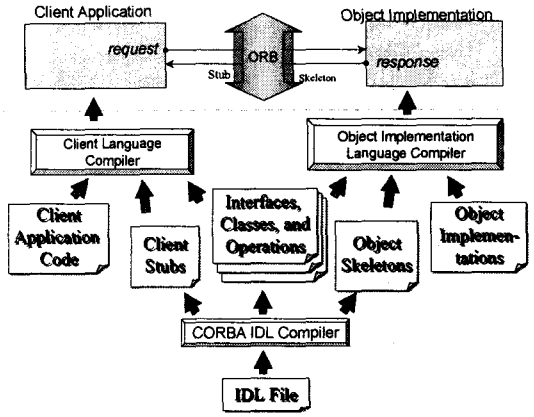

그림 6. IDL을 이용한 개발방법

즉, 이와 같은 IDL 언어를 사용하여 추상명세에서 정의된 Feature와 Geometry를 IDL로 정의하고, 이를 IDL컴파일러로 컴파일 하면, 스텁(Stub)과 스켈리톤(Skeleton), 그리고 기타 몇몇 클래스가 생성된다.

이때, IDL에서 JAVA로의 변환은 다음과 같은 규칙을 따른다.

표 2. IDL-JAVA변환규칙

IDL	JAVA	IDL	JAVA
constraint *in interface*	public static final	enum, struct, union,	class
constraint *Not in interface*	interface, public static final	interface	interface, helper class, holder class
sequence, array	array	exception	java class

3.3 OGC 추상명세에 대한 CORBA IDL 정의

☐ Feature 모델

OGC에서 정의한 추상명세에서 Feature 모델의 구조는 그림 7과 같다.

일반적으로 클라이언트 프로그램은 ContainerFeatureCollection을 통하여 접근하게 된다. ContainerFeatureCollection은 개념적으로

데이터베이스와 동일하며, Feature의 집합으로 구성된다.

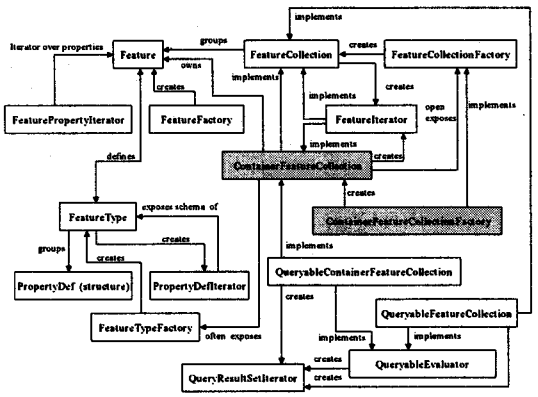

그림 7. Feature 모델

Feature는 Properties의 집합으로 Feature PropertyIterator에 의해서 반복된다. Feature Type은 각 property를 정의하는 Property Def의 집합이고, 이들은 ProfertyDefIterator를 통해서 접근하게 된다.

클라이언트 프로그램은 FeatureFactory를 통해서 새로운 Feature을 생성하는데 대부분 ContainerFeatureCollection으로 이 역할을 수행한다. FeatureTypeFactory는 클라이언트 프로그램이 새로운 FeatureTypes를 생성할 수 있도록 해준다. ContainerFeature Collection은 FeatureCollection의 특수한 형태이다.

□ Geometry 모델

Geometry는 지형공간정보에 대한 기하학적인 정보만을 갖는 객체모델 구조이다. Geometry는 다음과 같이 좌표의 차원에 따라 0차원에서 4차원으로 구분한다. 때에 따라 그 이상도 사용된다.

0-차원	1-차원	2-차원	3-차원	4-차원
points	curves	surfaces	solids	hyper solids

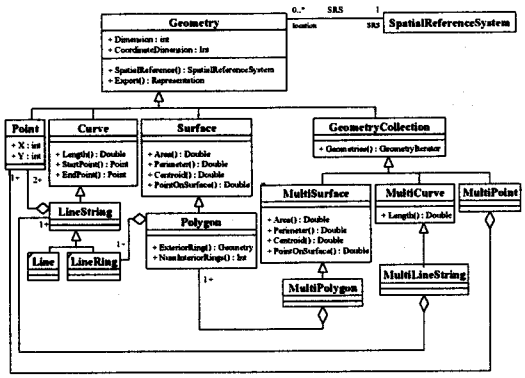

그림 8. Geometry 모델

Feature는 Geometry로 표현되는 spatial 데이터와 non-spatial 데이터의 집합으로 정의된다. 이에 대한 관계는 그림 9와 같다.

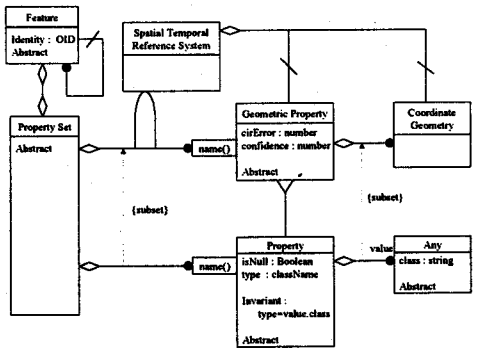

그림 9. Feature, Geometry간의 관계

OGIS추상명세에 대한 CORBA IDL은 표 3과 같다.

표 3. OpenGIS 구현명세 CORBA IDL 정의 중 Interface부분

```
module OGIS {
 //-----------------
 // Common structures
 //-----------------

 .....

 //--------------------
 // Well-known Structures
 //--------------------

 .....

 //--------------------
 // Feature Interface
 //--------------------
 interface Feature {};
 typedef sequence <Feature> FeatureSeq;
 interface FeaturePropertySetIterator {};
 interface FeatureFactory {};
 interface FeatureType {};
 interface FeatureTypeFactory {};
 interface PropertyDefIterator {};
 interface FeatureCollection : Feature {};
 interface FeatureCollectionFactory {};
 interface FeatureIterator  {};
 interface ContainerFeatureCollection : FeatureCollection,
                                        FeatureFactory {};
 interface ContainerFeatureCollectionFactory {};
 interface QueryEvaluator {};
 interface QueryableFeatureCollection : FeatureCollection,
                                        QueryEvaluator {};
 interface QueryableFeatureCollectionFactory {};
 interface QueryableContainerFeatureCollection:
           ContainerFeatureCollection, QueryEvaluator {};
 interface QueryableContainerFeatureCollectionFactory {};
 interface QueryResultSetMetaData;
 interface QueryResultSetIterator {};
 interface QueryResultSetMetaData {};

 //-----------------------------------
 // Spatial Reference Systems
 //-----------------------------------
 interface SpatialReferenceInfo {};
 interface Unit : SpatialReferenceInfo {};
 interface AngularUnit : Unit {};
 interface LinearUnit : Unit {};
 interface Ellipsoid : SpatialReferenceInfo {};
 interface HorizontalDatum : SpatialReferenceInfo {};
 interface PrimeMeridian : SpatialReferenceInfo {};
 interface SpatialReferenceSystem : SpatialReferenceInfo {};
 interface GeodeticSpatialReferenceSystem :
                        SpatialReferenceSystem {};
 interface GeographicCoordinateSystem :
                        GeodeticSpatialReferenceSystem {};
 interface Parameter : SpatialReferenceInfo {};
 typedef sequence<Parameter> ParameterSeq;
 interface ParameterList {};
 interface GeographicTransform : SpatialReferenceInfo {};
 interface Projection : SpatialReferenceInfo {};
 interface ProjectedCoordinateSystem :
                        GeodeticSpatialReferenceSystem {};
 interface SpatialReferenceSystemFactory {};
 interface SpatialReferenceComponentFactory {};

 //---------------------------------------
 // Geometry interface
 //---------------------------------------
 interface Geometry {};
 interface GeometryFactory {};
 interface GeometryCollection : Geometry {};
 interface GeometryIterator {};
 interface Point : Geometry {};
 interface PointFactory : GeometryFactory {};
 interface MultiPoint : GeometryCollection {};
 interface MultiPointFactory : GeometryFactory {};
 interface Curve : Geometry {};
 interface LineString : Curve {};
 interface LineStringFactory : GeometryFactory {};
 interface Ring : Curve {};
 interface LinearRing : Ring, LineString {};
 interface MultiCurve : GeometryCollection {};
 interface MultiLineString : MultiCurve {};
 interface MultiLineStringFactory : GeometryFactory {};
 interface Surface : Geometry {};
 interface Polygon : Surface {};
 interface LinearPolygon : Polygon {};
 interface LinearPolygonFactory : GeometryFactory {};
 interface MultiSurface : GeometryCollection {};
 interface MultiPolygon : MultiSurface {};
 interface MultiLinearPolygon : MultiPolygon {};
 interface MultiLinearPolygonFactory : GeometryFactory {};
}; // End OGIS Module
```

3.4 JAVA 스텁과 스켈리톤의 생성

VisiBroker의 IDL 컴파일러는 idl2java로 위 표 3에서 정의된 OGIS IDL을 컴파일하면 다음과 같은 스텁, 스켈리톤 등이 생성된다.

IDL : interface <*식별자*>

idl2java ⇓

public class <*식별자*>
public class <*식별자*>Helper
public class <*식별자*>Holder

public class _st_<*식별자*>

public class _<*식별자*>ImplBase
public class _example_<*식별자*>
public class _tie_<*식별자*>
public class <*식별자*>Operarions

- <***식별자>Helper*** : Struct, enum, union 등과 같은 객체에 대하여 stream으로의 쓰기와 읽기를 위한 메쏘드와 객체의 repository identifier를 되돌려주는 메쏘드를 지원한다.
- <***식별자>Holder*** : 파라미터의 전달을 위한 holder[2]를 제공한다.
- ***_st_<식별자>*** : 스텁 구현을 지원한다.
- ***_<식별자>ImplBase*** : 서버에서 객체구현을 위한 스켈리톤 코드이다.
- ***_tie_<식별자>*** : tie기법[3]을 이용하여 서버에서 객체구현시 사용된다.

---

2) JAVA의 경우 Call by Value만을 제공하기 때문에 VisiBrober에서 Call by Reference를 지원하기 위하여 만들어진 클래스이다.

3) 객체구현 클래스는 VisiBroker 스켈리톤 클래스로부터 상속을 받는다. 그러나, java의 경우 multiple class inheritance를 허용하지 않는다. Multiple class inheritance를 지원하기 위한 기법이 tie 기법이다.

○ *_<식별자>Operations* : tie기법을 이용하여 구현할 경우 함께 사용되는 클래스이다.

○ *_example_<식별자>* : 서버에서 객체구현에 채워넣을 수 있는 코드를 제공한다.

이와 같은 IDL에서 JAVA로의 변환을 표 3에서 정의된 IDL 중 Feature Interface 부분을 중심으로 살펴보면 다음과 같다.

먼저, OCG의 CORBA구현중 Feature에 대한 정의는 다음 표 4와 같다.

표 4. Featue Interface IDL 명세

```
interface Feature {
  exception InvalidParams {string why; };
  exception InvalidProperty {};
  exception InvalidValue {};
  exception PropertyNotSet {};
  exception RequiredProperty {};

  readonly attributes FeatureType Feature_type;

  Geometry get_geometry(in NVPairSeq geometry_context)
                raises (InvalidParams);
  boolean   property_exists(in Istring name)
                raises(InvalidProperty);
  any       get_property(in Istring name)
                raises(PropertyNotSet, InvalidProperty);
  void      set_property(in Istring name, in any value)
                raises(InvalidProperty, InvalidValue);
  void      delete_property(in Istring name)
                raises(PropertyNotSet, InvalidProperty,
                    RequiresProperty);
  NVPairSeq   get_property_sequence(in Unsigned long n);
  FeaturePropertySetIterator    get_property_iterator();
  void      distroy();
};
```

이를 컴파일 한 결과가 그림 10이다. 앞에서 살펴본 바와 같이 Feature에 대한 스텁, 스켈리톤 등이 생성된다. exception에 대하여 중첩된 package로 Feature Package가 생성된다[4]. 그리고, 각각 '*<식별자>*', '*<식별자>*Helper', 그리고 '*<식별자>*Holder' 클래스가 생성된다.

4) IDL의 경우 인터페이스내에서 중첩된 type정의를 허용하나, Java의 경우 이를 허용하지 않기 때문에 새로운 Package를 생성하여 해결한다.

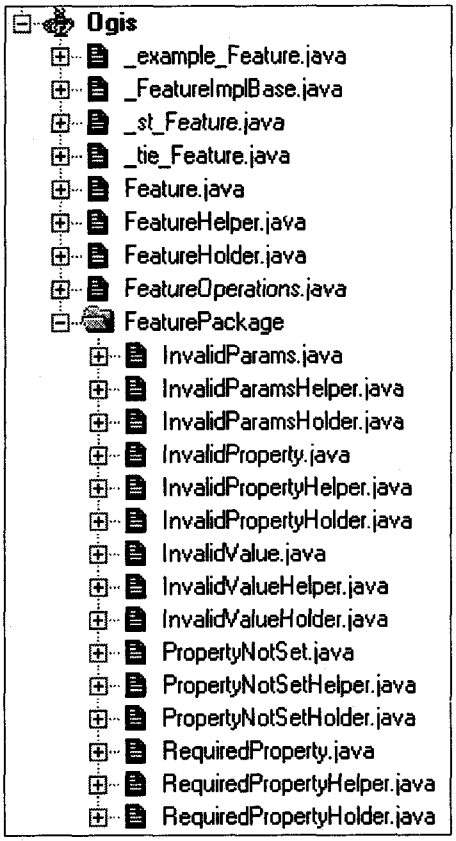

그림 10. IDL컴파일 결과중 Feature부분

3.5 서버 및 클라이언트의 구현

그림 5에서 제시된 분산객체GIS 구현모델에서 DB Service Processing Provider와 Basic Function Package에 대해서 각각 그림 10에서 생성된 Feature 스켈리톤과 스텁을 중심으로 구현하는 방안을 살펴보면 다음과 같다.

3.5.1 DB Service Processing Provider의 구현

DB Service Processing Provider는 _example_Feature.java에 각각의 매쏘드 각각에 원하는 로직의 프로그램을 구현하고, 구현객체를 생성한다.

표 5. _example_Feature.java 소스코드

```
package OGIS;

public class _example_Feature extends OGIS._FeatureImplBase {
  public _example_Feature(java.lang.String name) {
    super(name);
  }
  public _example_Feature() {
    super();
  }

  public OGIS.Geometry get_geometry(OGIS.NVPair[] geometry_context)
                                throws OGIS.FeaturePackage.InvalidParams
  {
    // 구현부분
    return null;
  }

  public boolean property_exists(java.lang.String name)
                                throws OGIS.FeaturePackage.PropertyNotSet,
                                       OGIS.FeaturePackage.InvalidProperty
  {
    // 구현부분
    return false;
  }

  public org.omg.CORBA.Any get_property(java.lang.String name)
                                throws OGIS.FeaturePackage.PropertyNotSet,
                                       OGIS.FeaturePackage.InvalidProperty
  {
    // 구현부분
    return null;
  }

  public void set_property(java.lang.String name,org.omg.CORBA.Any value)
                                throws OGIS.FeaturePackage.InvalidProperty,
                                       OGIS.FeaturePackage.InvalidValue
  {
    // 구현부분
  }

  public void delete_property(java.lang.String name)
                                throws OGIS.FeaturePackage.PropertyNotSet,
                                       OGIS.FeaturePackage.InvalidProperty,
                                       OGIS.FeaturePackage.RequiredProperty
  {
    // 구현부분
  }

  public OGIS.NVPair[] get_property_sequence(int n)
  {
    // 구현부분
    return null;
  }

  public OGIS.FeaturePropertySetIterator get_property_iterator()
  {
    // 구현부분
    return null;
  }

  public void destroy()
  {
    // 구현부분
  }

  public OGIS.FeatureType feature_type()
  {
    // 구현부분
    return null;
  }
}
```

그리고, 생성된 구현객체를 ORB에 등록하기 위한 main 함수를 작성한다.

main함수에서는 ORB를 초기화하고, 클라이언트에게 제공되는 객체의 활성화/비활성화를 위하여 객체구현에서 사용되는 BOA(Basic Object Adapter)를 초기화하고, 새롭게 생성된 객체를 BOA에 등록하고, 클라이언트의 요청을 기다리게 된다. 이에 대한 클래스를 FeatureServer라 할 때 그 구조는 다음과 같다.

표 6. ORB로의 구현객체 등록클래스 작성

```
.....

public class FeatureServer {

  public static void main (String[] args) {
    try {
        // ORB 초기화
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();

        // BOA 초기화
        org.omg.CORBA.BOA boa = orb.BOA_init();

        // Feature 객체 생성
        Feature feature = new Feature("fname");
        // 새롭게 생성된 Feature 객체를 BOA에 등록한다.
        boa.obj_is_ready(feature);

        // 클라이언트의 요청을 기다린다.
        boa.impl_is_ready();

    } catch(CORBA.SystemException e) {
        // 예외처리
    }
  }
}
```

3.5.2 Basic Function Package의 구현

Basic Function Package 중 Feature에 대한 부분을 살펴보면 다음과 같다.

표 7. Feature Client 부분

```
.....

public class FeatureClient {

  public static void main (String[] args) {
    try {
        CORBA_Any = propertyValue;

        // ORB 초기화
        org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init();
        // 서버의 "fname"을 갖는 Feature 객체에 대한 객체 참조자를
           얻어옴
        Ogis.Feature featureRef = Ogis.FeatureHelper.bind("fname");
        Ogis.FeatureType featureTypeRef =
                              featureRef.get_feature_type();
        Ogis.PropertyDefSeq propertySet =
                              featureTypeRef.get_property_defs();
        // 각 Property에 대하여 작업을 수행함
        for (i=0; i < propertySet.length ; i++ )
        {
            propertyValue =
                featureRef.get_property((propertySet.get(i)).name);
        }

    } catch(Ogis.FeaturePackage.InvalidProperty e) {
        // 예외처리
    } catch(Ogis.FeaturePackage.PropertyNotSet e) {
        // 예외처리
    } catch(CORBA.SystemException e) {
        // 예외처리
    }
  }
}
```

먼저 ORB를 초기화하고, 서버에서 정의된

Feature 객체의 이름을 통하여 Feature 객체에 대한 객체참조자를 얻어온다. 얻어진 객체참조자를 이용하여 다른 Java객체와 동일한 방법으로 객체를 이용하여 원하는 로직에 맞게 구현을 하게 된다.

4. 향후추진방안 및 결론

이상과 같이 OGC의 CORBA 구현명세를 기반으로 하여 분산객체GIS 구현모델을 설정하고, JAVA와 VisiBroker를 이용하여 서버에서의 구현과 클라이언트에서의 구현 방안을 제시하였다 CORBA는 이종의 시스템, 운영체제 등과 같은 환경 하에서 프로그래밍언어에 독립적인 프로그램을 작성하고, 실행코드 수준에서의 시스템 통합을 구현할 수 있도록 해준다. 이와 같은 CORBA의 기능은 JAVA의 이종 시스템에서의 원시코드 및 실행코드 수준에서의 호환성과 유사하다. 이러한 이종 시스템에서의 호환성에 대한 유사성은 OMG에서 IDL에 기초하지 않고 CORBA와 JAVA 연동을 위한 표준화를 추진하는 것을 볼 때 더욱 확연해 진다.

또한, JAVA의 Applet과 Servlet은 Web과의 통합적인 측면에서 아주 유용하다. 이러한 유용성과 JAVA와 CORBA의 호환성을 감안할 때, 차후 CORBA를 기반으로 하는 분산객체GIS 구축시 JAVA의 활용이 늘어나게 될 것으로 보인다. 특히 분산객체GIS 클라이언트의 경우, 범용 사용자인터페이스로 정착되고 있는 웹과의 통합을 고려할때 JAVA로의 구현은 필수적이라 할 수 있다.

이와 같이 뛰어난 호환성을 갖고 있는 JAVA를 이용하여 OGC의 CORBA구현명세를 바탕으로 분산객체GIS 구현모델을 제시함으로써 표준화된 시스템의 구현이 가능할 것이며, 세부적인 Package에 대한 기능 및 데이터정의가 추진될 경우 전체 시스템적인 차원에서의 상호호환성을 이룩할 수 있을 것이다.

참고문헌

[OGP99-016] OpenGIS Consortium, Inc. , OpenGIS Project Document 99-016, Revisions to the Query Model for OpenGIS Simle Features Specification for CORBA, 1999.

[OGA99] Open GIS Consortium, Inc., The OpenGIS Abstract Specification Model, Version 4, 1999

[OGO99] Open GIS Consortium, Inc., OpenGIS Simple Features Specification for OLE/ COM Revision 1.1, 1999

[OGS99] Open GIS Consortium, Inc., OpenGIS Simple Features Specification for SQL Revision 1.1, 1999

[OGC98] Open GIS Consortium, Inc., OpenGIS Simple Features Specification for CORBA, Revision 1.0, 1998

[VBJ34] VisiBroker for Java - Reference Version 3.4, Insprise.

[VBJ34] VisiBroker for Java - Programmer's Guide Version 3.4, Insprise

[박98] 박재현, 코아코바, 영한출판사, 1998

[정98] 정준원, 여찬기 등, 자바를 이용한 데이터베이스 연동웹서비스 기술보고서, 한국전산원, 1998.

손 진 락

1996년 건국대학교 대학원 전자계산학과 졸업 (공학석사)

1996년~현재 한국전산원 국가정보화센터 주임연구원

관심분야 : 분산객체기술, 에이전트, GIS 등

정 준 원

1997년 홍익대학교 대학원 전기제어공학과 졸업(공학석사)

1997년~현재 한국전산원 정보화평가분석단 주임연구원

관심분야 : 분산객체기술, 진화알고리즘, JAVA 등

진 희 채

1995년 서울대학교 대학원 산업공학과 졸업 (공학박사)

1995년~현재 한국전산원 정보화평가분석단 선임연구원

관심분야 : 개방형 GIS, 최적화 알고리즘 등