# OpenGIS Conforming Map-Feature Server Implementation Specifications in Component-based Distributed Systems

Shifeng Zhang* and Steve Goddard

Department of Computer Science and Engineering
University of Nebraska-Lincoln, Lincoln, Nebraska 68588-0115, U.S.A

**Abstract.** Geo-data and geo-service interoperations are important to Geographic Information System (GIS) users. To meet such needs, standard specifications are required for geo-data and geo-services. The OpenGIS Consortium (OGC) defined abstract specifications and partial implementation specifications, which can make diverse geo-data and geo-services accessible to conforming applications. However the implementation specifications are not complete, particularly in the areas of interactive map servers and feature servers. Currently available map server and feature server implementation specifications are only for HTTP-based Web environments; the implementation specifications for non-HTTP environments, such as CORBA, are still undefined.

The National Agricultural Decision Support System (NADSS) is a component-based distributed GIS, which uses CORBA as the distributed computing platform. We extend the OGC Web Map Server (WMS) implementation specification and Web Feature Server (WFS) implementation specification to CORBA-based components. Similar to WMS and WFS implementation specifications, clients conforming to our proposed implementation specification can interact with a map-feature server in CORBA environments. Moreover, the transmitted geo-data conform to the standard OpenGIS simple feature and grid/coverage specifications. To implement our proposed specifications, we used GRASS, a free open-source GIS, as the basis for the CORBA map-feature server. GRASS is a traditional command-oriented GIS, which is not suitable as a distributed map server or feature server. We transformed GRASS into a CORBA map-feature server in a distributed computing environment using our proposed specification. To evaluate the performance of our implementation, we compared the OpenGIS conforming map-feature server implementation with a non-standard, pure data structure map-feature server implementation. The evaluation results show the standard OpenGIS-based interoperability comes at a cost in terms of performance.

*Keywords:* OpenGIS, Map Server, Feature Server, CORBA, GRASS.

## 1. Introduction

Traditional Geographic Information System (GIS) software provide raster and vector geo-data presentation and processing services to GIS users. However, the geo-data structure and the geo-service interfaces are often proprietary. For example, vector data can be represented as shape files in ArcInfo while as a collection of directory-based files in GRASS. Such heterogeneity makes GIS software interoperation difficult. To solve this problem, the OpenGIS Consortium (OGC) proposed standard specifications, including geo-data presentation and geo-service interfaces. To standardize the geo-data format, Simple Feature (SF) (OGC-SF, 1998) and Grid/Coverage (GC) (OGC-GC, 2001) implementation specifications are proposed for vector data and raster data. To standardize the geo-services interaction, Web Mapping Server (WMS) (OGC-WMS, 2002) and Web Feature Server (WFS) (OGC-WFS, 2002) implementation specifications were proposed for map (raster data) presentation and features (vector data) manipulation via the Web. Given the diversity of distributed computing environments, WMS and WFS implementation specifications are not enough for today's GIS applications; map presentation and feature manipulation specifications should also be available in other distributed computing environments, such as Common Object Request Broker Architecture (CORBA) and Distributed Component Object Model (DCOM).

Many of today's GIS applications are component-based applications. They rely on CORBA or DCOM middleware. The National Agricultural Decision Support System (NADSS) is such an example. NADSS is being developed with the Risk Management Agency of the United States Department of Agriculture (Goddard et al., 2002).

* Corresponding author: shzhang@cse.unl.edu

The initial focus of the NADSS project is to improve the quality and accessibility of drought related knowledge, information, and spatial analysis for drought risk management. CORBA technology is used as the distributed computing environment. Since we want to make raster and vector geo-data in NADSS available to outside GIS applications, standard interfaces for the map server and feature server are necessary. This paper introduces the proposed map-feature server implementation specification for CORBA based on the OpenGIS WMS and WFS implementation specifications. The rest of this paper first introduces the OpenGIS WMS and WFS implementation specifications in Section 2, and then describes our proposed client-push map-feature server for CORBA in Section 3. Section 4 introduces the implemented map-feature server based on GRASS and performance evaluation with a non-OpenGIS conforming implementation. Section 5 presents our conclusions.

## 2. Background

This section introduces background knowledge for map servers and feature servers. Section 2.1 introduces the concepts of map servers and feature servers; the implementation specifications for WMS and WFS from OGC are introduced in Section 2.2; and the advantages of similar implementation specifications in other distributed computing platforms are described in Section 2.3.

### 2.1. Map Servers and Feature Servers in GIS

In GIS, a map server organizes and publishes maps (raster data) while a feature server organizes and publishes geographic simple features (vector data). They serve as brokers to map and feature databases. A rendered map is the final result from a map server while a simple feature is returned from a feature server. In (Cuthbert, 1997), Cuthbert divided the geo-spatial data presentation into four processes:

1. The selection of geo-spatial data to be displayed;
2. The generation of display elements from the selected geo-spatial data;
3. The rendering of display elements into a rendered map;
4. The display of the rendered map to the user.

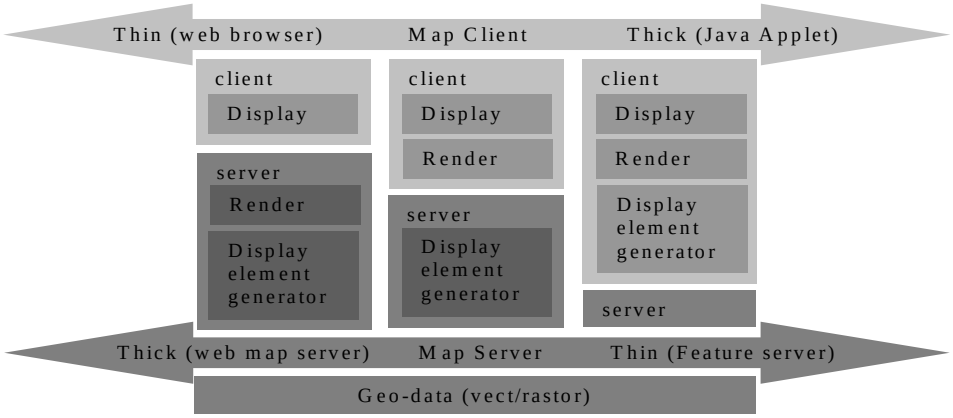


**Figure 1**. Three kinds of map servers and corresponding map clients.

As Figure 1 shows, the interfaces between the map client and the map server can be dynamic; it depends on what kind of geo-data is transmitted between the client and the server. If the map client only has display capabilities, pixel data (raster data) will be transmitted to the client. All standard Web browsers have such a capability. If the map client has map-rendering and display element generator capabilities, simple features (the vector data) can also be transmitted to the map client from the feature server. Compared with pixel data transmission, the advantage is that it reduces network traffic by transmitting vector geo-data only once, especially for the *zoom in/zoom out* and *pan* operations in the client side. The disadvantage is that the process requirements for the vector data map client are much higher than those for the pixel data map client. A typical vector data map client needs a Java applet, Active-X component or plug-in to manipulate the vector data.

### 2.2. OGC WMS Implementation Specification and WFS Implementation Specification

Figure 1 indicates a situation in which only pixel data are transmitted if the map client is a standard Web browser (e.g., IE or Netscape). The OGC WMS implementation specification is such an example that specifies the HTTP-based interactive interface between standard Web browsers and Web servers. The advantages of WMS are as follows.

- A well-known port is used. The HTTP port is open for almost every server in the internet; almost no firewalls will block HTTP port transportation.
- Standard and powerful metadata description. XML is used to describe the meta-information (capability) of the WMS; XML conveys more semantic information than HTML.
- Standard clients already exist. A Web browser is the most common Web client tool for WMS.

Unlike WMS, WFS transmits the vector data in the OGC simple feature format described by Geographic Markup Language (GML). The feature client can query, create, modify and delete features in the feature server through the HTTP protocol. The advantages are the same as those for WMS, but currently standard Web browsers can only display the text feature information with GML. There is no accepted standard for displaying vector data on client browsers (though recently, scalable vector graphics have been adopted as the standard by the WWW Consortium (Andersson, 2003)). Figure 2 shows the architecture for WMS and WFS.

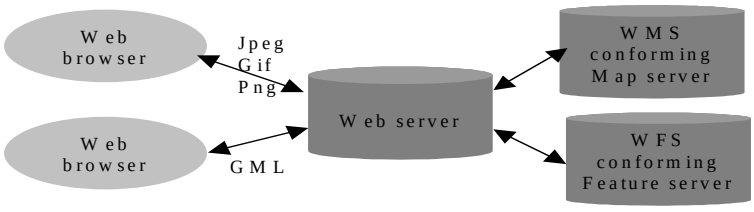


**Figure 2.** Web map server and Web feature server architecture.

### 2.3. Map Server and Feature Server Implementation Specifications for non-HTTP Environments

The OGC WMS and WFS implementation specifications use HTTP Get/Post methods to present the map and simple feature data via the Web. However, standard specifications for map and feature servers are still undefined for other distributed computing environments. The needs for such specifications include component interaction, server-side efficiency and client-side extensibility. The following paragraphs explain each of these needs.

- Component interaction. A component is a fairly independent computing unit with exposed interfaces; components can collaborate together to fulfill a special application requirement. The number of component-based GIS applications is increasing, e.g., (Kumar, 1997; Tsou, 1998). Component-based middleware includes CORBA, DCOM, and Java. It is necessary to design map and feature server implementation specifications for these middleware environments.
- Server-Side efficiency. As Figure 2 shows, the communication between the map client and the map server is not direct. The Web server acts as a broker between them. In fact, most current interactions between the Web server and the map/feature server use CGI, which are process-based applications. This is inefficient on the server side. The direct communication between the map client and map server in CORBA or DCOM environments would be more efficient than WMS and WFS.
- Client-Side Extensibility. In GIS software, image or feature manipulation functions, such as zoom or pan operations, are necessary. It is not enough to merely display images or GML-based simple features in a standard Web browser. A thick client, typically in CORBA or DCOM environments, can provide more manipulation capabilities. Geotools (Geotools, 2003) and Geo Viewer (Geoviewer, 2003) are examples of such thick clients.

With the map-feature server implementation specification proposed in the next section, these needs can be satisfied for a CORBA distributed computing environment.

## 3. OpenGIS Conforming Map-Feature Server for CORBA

Many GIS applications are based on CORBA, DCOM, and Java middleware (Kumar, 1997; Koschel, 1997; Tsou,

1998; Geotools, 2003; Geoviewer, 2003). A lack of standard map and feature server implementation specifications in CORBA, DCOM, and Java environments makes geo-data and geo-service interoperation difficult. WMS and WFS implementation specifications provide standard map and feature access methods through HTTP. However, limited by the XML language, which is used to define the XML DTD for WMS and WFS implementation specifications, it is difficult to implement the object concept in WMS and WFS. Moreover, in most of today's WMS and WFS implementations, CGI-based implementations are adopted. CGI is process based and less efficient than a thread-based implementation. Using CORBA or DCOM technology, object-oriented concepts can be used to design map and feature servers, which have greater efficiency and flexibility than Web-based servers. Section 3.1 introduces the interface design of the proposed map-feature server from the XML Document Type Definition (DTD) of the WMS and WFS implementation specifications. Section 3.2 introduces the client-push map server concept in the proposed map-feature server, and Section 3.3 introduces the relationship between the proposed map-feature server implementation specification, the OGC Simple Feature (SF) implementation specification and the OGC Grid/Coverage (GC) implementation specification. The full interface definition is given in Appendix 1.

### 3.1. The Definition of the CORBA Map-Feature Server Interface

WMS and WFS implementation specifications are the existing standards for map presentation and feature manipulation via the Web. Thus it makes sense to use them as the basis for specifications in other distributed computing platforms. We extend them to be the implementation specifications for the CORBA distributed environment; in addition to that, we add some new features described in Sections 3.2 and 3.3.

To simplify the proposed implementation specification, we combined the map server and feature server implementation specification into one map-feature server implementation specification. The client side can combine vector data and raster data under one map-feature server implementation. To create a CORBA map-feature server implementation specification, it is necessary to translate the WMS and WFS XML DTD to interfaces defined by CORBA Interface Definition Language (IDL). Fortunately, the translation is straightforward. For example, the two operations from WMS, *getCapabilites()* and *getMap()*, and one operation from WFS, *getFeature()*, are directly defined in the IDL. Operation *getCapabilities()* returns the available map layers and simple features information for the map-feature server. Operation *getMap()* returns the requested map layers, and *getFeature()* returns the required simple features. Creating the corresponding data structures based on WMS or WFS's XML DTD is straightforward too. Below is one example which shows the original data structure *LatLongBoundingBOX* in WMS's XML DTD and the translated data structure in IDL:

```
<! ATTLIST LatLongBoundingBox          Struct LatLongBoundingBox {
        minx CDATA                          Double minx;
        miny CDATA                          Double miny;
        maxx CDATA                          Double maxx;
        maxy CDATA                          Double maxy;
>                                       }
```

**Figure 3.** Data structure definition in the XML DTD and the CORBA IDL.

### 3.2. Client-Push Map-Feature Server

The WMS implementation specification defines a client-pull map server, in which clients can only get maps from the map server based on the map server's capability. A client-pull map server is a "*read only*" map server; clients cannot provide their own data to generate the desired map. The OGC's Styled Layer Descriptor implementation specification lets users define the presentation style of the map layer (OGC-SLD, 2001), which allows the users to have "*write*" capability to create new presentation styles for a map layer, but users still cannot generate new layers with their data.

Client-push map servers, on the other hand, are more active than client-pull map servers. With a client-push map server, users can make use of the GIS software in the map servers to generate their desired maps. Considering that current commercial GIS software is expensive, and they require high-performance machines for map generation, a client-push map server is useful for users. For example, in NADSS, the map server is used to generate maps more than to query maps: users provide site-based data to the map server and get an interpolation map generated from their data. To implement such a client-push map server, we add a new operation *generateMap()* to provide the map generation service. The new operation uses the interpolation functions of the underlying map server to generate the

map. Also, some new basic service elements are added to the proposed implementation specification to support map generation. First, a new layer attribute called "user_create" is added in the service element "layer". It indicates whether such a layer can be generated based on the user's input data. Second, a new data structure is defined to hold the input point data.

### 3.3. OpenGIS Geo-data Presentation in the CORBA Map-Feature Server

The OGC standardized geo-data presentation with the SF implementation specification and the GC implementation specification. The SF implementation specification defines the vector data interfaces and GC implementation specification defines the raster data interfaces. Limited by the HTTP Get/Post method, the return value of operation *getMap()*in the WMS implementation specification is a http address to a gif, jpeg or png files; Similarly, the return value of operation *getFeature()* in the WFS implementation specification is the feature presentation in GML format. To conform to the OGC's geo-data specifications, we adopt the simple feature as the returned vector data format and the grid coverage as the returned raster data format from the map-feature server. By that we assure the transmitted geo-data are standard OpenGIS conforming geo-data. As depicted in Figure 4, the proposed CORBA map-feature server implementation specification is built on top of the SF and GC implementation specification.

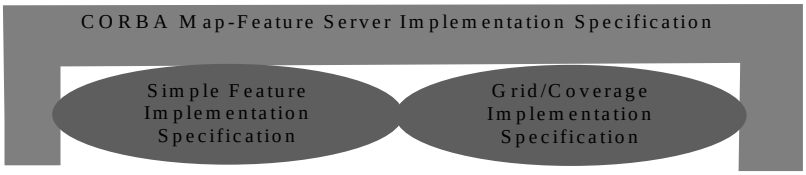


**Figure 4.** Map-feature server built based on SF and GC implemented specifications.

## 4. CORBA Map-Feature Server in NADSS

NADSS is a component-based distributed system. It provides geo-related data in the form of climate data (e.g., temperature and precipitation) for agricultural models; information in the form of drought indices; and knowledge in the form of exposure analysis. Two methods are used to make the data, information and knowledge available to users. First, Web-based interfaces are provided by which users can access and download data, information and knowledge from the Web site. Second, a CORBA-based Java applet client is provided. Users automatically download the applet from the NADSS Web site to access the data, information and knowledge. By providing the OpenGIS conforming map-feature server, conforming clients can have direct access to the simple feature and grid/coverage data. Section 4.1 introduces the architecture and the implementation with a GIS application. The evaluation of the implementation is presented in Section 4.2.

### 4.1. Map-Feature Server Architecture and Implementation with GRASS

Figure 5 shows the architecture we have used to transform a non-interoperable geo-spatial decision support system into an OpenGIS conforming system. By adding the map-feature server that conforms to the proposed implementation specification, an outside client can get OpenGIS conforming simple features and grid/coverage data from the non-OpenGIS conforming data sources.

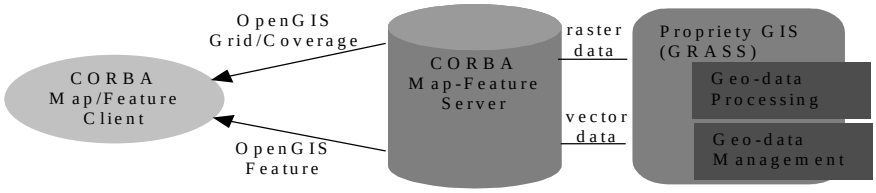


**Figure 5.** OpenGIS conforming map-feature server in NADSS

In NADSS, GRASS, a free open-source GIS, is the underlying GIS software. GRASS provides powerful raster/vector data processing functions. However, it is a traditional command-oriented GIS (Neteler, 2002). Unlike

ArcInfo, which can provide an internet-based solution (ESRI, 2002), GRASS currently lacks the capability to expose its geo-processing functions and geo-data in a distributed environment. GRASSLinks is a popular method to expose GRASS functions and geo-data through the Web (Huse, 1995). Since GRASSLinks is a CGI-based method, it is neither efficient nor flexible because of the drawbacks of CGI. However, its popularity demonstrates the need to adapt GRASS to a distributed environment.

It is necessary to change GRASS from its current command-oriented style to a component-oriented style, which can provide the GRASS functions through an object-oriented style in a distributed environment. In NADSS, we separate the GRASS functions into two categories: geo-data processing and geo-data management. Within the geo-data processing part, GRASS map generating functions are provided as a client-push map server; while within the geo-data management part, GRASS vector data is served as simple features and raster data as grid/coverage.

Koschel did similar work by wrapping GRASS with CORBA objects (Koschel, 1997). The difference is that he used GRASS only as a readable GIS server while in NADSS we make use not only of the geo-data management functions but also of the geo-data processing (interpolation) functions. More importantly, in (Koschel, 1997) the interfaces are proprietary, so it is impossible for OpenGIS conforming clients to access the geo-data. While in NADSS, simple features with the following geometries are implemented: *Point*, *LineString*, *LinearRing* and *LinearPolygon*. We also partially implemented the Grid/Coverage interface; so a conforming client can access the standard simple features and the grid/coverage data from the map-feature server.

Figure 6 is the screen snap-shot of a client that conforms to our proposed specification. It displays the OGC conforming simple feature data (roads as black lines) and grid-coverage data (basemap) together; the data are served from GRASS vector data and raster data.

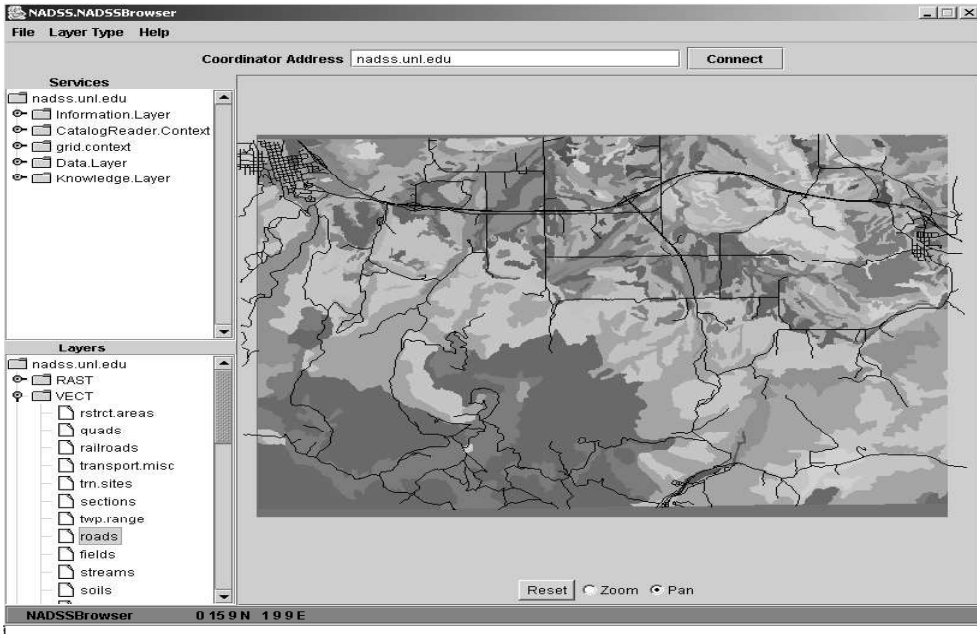


**Figure 6.** Screen snapshot of an OGC conforming client.

### 4.2. Performance Evaluation of the CORBA Map-Feature Server

The OGC's implementation specifications make different GIS software interactions possible. However, interoperability comes at a price in terms of performance, especially with the limitations of CORBA. Currently CORBA's specification does not support object serialization. It cannot transmit an object by its value instead of its reference. Since OGC conforming clients can only get references to the OGC simple features and grid/coverage objects, they need further communication with the map-feature server to get the geo-data. This communication is sometimes heavy, which may cause poor performance. To evaluate the performance of our implementation, the transfer time between simple feature objects and the pure data structure from the map-feature server to the client is compared. We tested the *Point* simple feature object in the OpenGIS conforming map-feature server implementation and the point data structure *{double x, double y}* in a non-OpenGIS conforming implementation. The unit of transfer time is in micro-seconds and the average transfer time for 25, 48, and 90 points are shown in Figure 7. We found that

the transfer time for a simple feature is nearly 5 times than that of the data structure. Such performance limitations will be reduced when Object Management Group proposes a CORBA specification that supports objects transmitted by value.

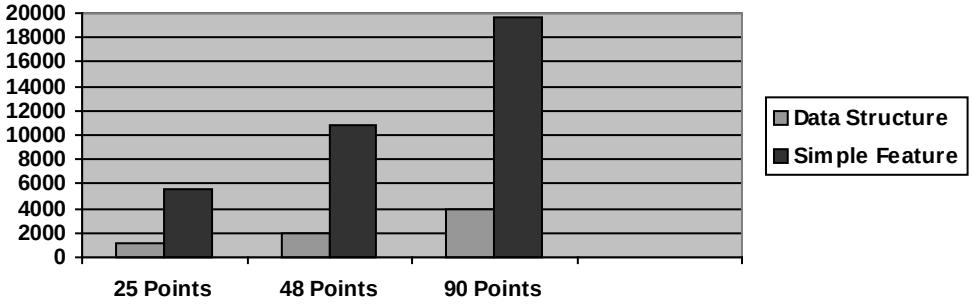


**Figure 7.** The evaluation of the CORBA map-feature server (the y-axis time units are ms).

## 5. Conclusion

The OpenGIS related implementation specifications are still under development. The OGC provides the SF and GC implementation specifications to present standard vector and raster geo-data. However, the map server and feature server implementation specifications are undefined for some distributed computing environments. OGC WMS and WFS implementation specifications are only available for HTTP-based environments. WMS "specifies implementation and use of those WMS operations in the HTTP Distributed Computing Platform (DCP). Future versions may apply to other DCPs" (OGC-WMS, 2002). Many GIS applications in other distributed environments, like CORBA, DCOM, and Java, need similar implementation specifications to interoperate with each other. This work proposes a CORBA map-feature server implementation specification, which can also be extended to DCOM and Java environments. It provides map and feature services similar to WMS and WFS for CORBA environments. Moreover, it also allows users to generate maps with their own data. Unlike WMS and WFS, which did not adopt the GC and SF implementation specifications, the geo-data in the proposed map-feature server implementation specification conform to the SF and GC implementation specification.

To implement the proposed implementation specification, we used GRASS as the underlying GIS server to provide the geo-data management and geo-data processing. Traditional GIS software, like GRASS, are command-oriented, and usually unsuitable for use in a component-based distributed computing environment. This limitation has prevented their wide usage in today's distributed GIS applications. We wrapped GRASS within a CORBA component to create map-feature server based on our proposed specification; and to the best of our knowledge, transforming GRASS to OpenGIS conforming CORBA objects has never been done before.

**Acknowledgments.** This work was supported, in part, by a grant from NSF (EIA-0091530) and a cooperative agreement with USADA FCIC/RMA (2IE08310228).

## References

Andersson, O., et al. (2003). Scalable Vector Graphics (SVG) Version 1.1. http://www.w3.org/TR/SVG11 (accessed by Apr 13, 2003).

Cuthbert, A. OpenGIS Consortium Inc. (1997). User Interaction with Geospatial data, OGC Document Number 98-060.

ESRI. (2002). ArcIMS 4 Architecture and Functionality, An ESRI White Paper.

Goddard, S., Zhang, S., Waltman, W., Lytle, D. and Anthony, S. (2002). A Software Architecture for Distributed Geospatial Decision Support Systems, Proc of 2002 national conference for digital government research, Los Angeles, CA. pp 45-52

Geotools. Open Source Mapping Toolkit. http://geotools.sourceforge.net/ (accessed Apr 13, 2003).

Geo Viewer. http://elib.cs.berkeley.edu/gis/ (accessed Apr 13, 2003).

Huse, S. (1995). GRASSLinks: A New Model for Spatial Information Access in Environmental Planning. Ph.D. Dissertation, Department of Landscape Architecture, University of California, Berkeley.

Koschel, A., Wiesel, J. (1997). GIS Operations under CORBA, http://www.lfu.baden-

wuerttemberg.de/lfu/uis/globus_direkt/globus3/223-fzic/gl3.html. (accessed Apr 13, 2003)
Kumar, K., Sinha, P., Bhatt, P.C.P. (1997). DoGIS: A Distributed and object oriented GIS. Proc. of the Seventh International Symposium on Spatial Data Handing, Netherlands. pp 263-275.
Neteler, M., Mitasova, H. (2002). Open Source GIS: A GRASS GIS Approach. Kluwer Academic Publisher. ISBN: 1-4020-7088-8.
OpenGIS Consortium Inc. (2001). OpenGIS Grid Coverage Implementation Specification For CORBA, OGC Project Document Number 01-004.
OpenGIS Consortium Inc. (1998). OpenGIS Simple Features Implementation Specification For CORBA.
OpenGIS Consortium Inc. (2001). OpenGIS Styled Layer Descriptor Draft Candidate Implementation Specification, OGC Documet Number 01-028.
OpenGIS Consortium Inc. (2002). Web Map Service Implementation Specification, OGC Document Number 01-068r3.
OpenGIS Consortium Inc. (2002). Web Map Service Implementation Specification, OGC Document Number 02-058.
Tsou, M-H., Buttenfield, B.P. (1998). Client/Server Components and Metadata Objects for Distributed Geographic Information Servers-GIServices, Proc. of GIS/LIS'98, Fort Worth, TX.

**Appendix 1.**

```
//Web location:
// http://nadss.unl.edu/ogc-map-feature-server.idl
//
//Proposed Map-Feature Server Implementation
Specification
//
#include "ogis.idl"
#include "ogc_gc.idl"

module mapfeatureserver {

        typedef sequence <string>  StringSeq;
        typedef sequence <double>  DoubleSeq;

        //at present it only have one input format here: SVP
= SiteValuePair
        enum inputformat {SVP};
        // this data struct contains input point data to
interpolate a raster layer
        struct SiteValuePair {
          string siteid;
          float value;
          double x;
          double y;
        };
        typedef sequence <SiteValuePair> SVPairSeq;


        //wms related data structure
        struct OnlineResource {
                string xmlns_xlink;
                string xlink_type;
                string xlink_href;
        };

        struct ContactPersonPrimary {
                string ContactPerson;
                string ContactOrganization;
        };

struct ContactAddress {
        string AddresType;
        string Address;
        string City;
        string StateOrProvince;
        string PostCode;
        string Country;
};

struct ContactInformation {
        ContactPersonPrimary cp;
        ContactAddress ca;
        string ContactVoiceTelephone;
        string ContactFacsimileTelephone;
        string ContactElectronicMailAddress;
};

struct BoundaryBox {
        double minx;
        double miny;
        double maxx;
        double maxy;
};

struct LatLonBoundaryBox {
        double minx;
        double miny;
        double maxx;
        double maxy;
};

struct SRSBoundaryBox {
        string SRS;
        double minx;
        double miny;
        double maxx;
        double maxy;
        double resx;
        double resy;
```

```
        };

        struct GetMapRequest {
                string version;
                StringSeq layers;
                StringSeq styles;
                string     srs;
                BoundaryBox bbox;
                long     width;
                long     height;
                string   format;
                boolean          transparent;
        };

        struct GetLayerRequest {
                string version;
                string layer;
                StringSeq styles;
                string     srs;
                BoundaryBox bbox;
                long     width;
                long     height;
                string   format;
                boolean          transparent;
        };

        //capability begin
        struct GetCapabilitiesRequest {
                string version;
                //we omit the service and request
        };


        struct Service {
                string Name;
                string Title;
                string Abs;
                StringSeq KeywordList;
                OnlineResource onliners;
                ContactInformation ci;

        };

        enum DCPType {CORBA, DCOM, JAVA};
        enum Operation {GetCapabilities, GetMap,
GetFeature, GenerateMap};
        struct RequestElement {
                Operation request_name;
                StringSeq Format;
                DCPType dcptype;
        };
        typedef sequence<RequestElement> Request;

        struct layer_attributes {
                boolean queryable;
                string cascades;
                boolean opaque;
                boolean noSubsets;
                //new push mode map server attributes
                boolean usercreate;
                //new push mode map server attributes the
inputformat should be well-known data structure
                inputformat infmt;
                double fixedWidth;
                double fixedHeight;
        };

        struct layers {
                layer_attributes la;
                string Name;
                string Title;
                string Abs;
                StringSeq KeywordList;
                string SRS; //from well-know text
                LatLonBoundaryBox llbb;
                SRSBoundaryBox srsbb;
                //we omit the rest part for its non-important
                //struct Dimension d;
                //struct Extent e;
                //struct Attributions a;
                //we also omit the style related part here
                sequence<layers> LaySeq;
        };

        typedef sequence<layers> LayerSeq;

        struct Capability {
                Request req;
                //we do not need exception in the data
structure
                //we omit the VendorSpecificaCapabilities
in the data structure
                //we omit the UserDefineddSymbolization
in the data structure
                LayerSeq l;
        };

        struct MS_Capabilities {
                Service s;
                Capability c;
        };
        //capability end

        interface MapFeatureServer{
                MS_Capabilities getCapabilities(in
GetCapabilitiesRequest psin);
                boolean generateMap(in GetMapRequest
gmr, in SVPairSeq svps);
                gc::GC_GridCoverage getMap(in
GetLayerRequest gmr);
                OGIS::FeatureCollection getFeature(in
GetLayerRequest gmr);
        };

};
```

**Shifeng Zhang** is a Ph.D. candidate in Computer Science and Engineering at the University of Nebraska-Lincoln, USA. He received his master's degree in Computer Science from East-China Institute of Computer Technology. His current research interests include distributed software engineering and GIS.

**Steve Goddard** is an assistant professor of Computer Science and Engineering at the University of Nebraska-Lincoln, USA. He received his Ph.D. in Computer Science from the University of North Carolina-Chapel Hill, USA. His research interests lie in the broad areas of real-time and distributed systems with an emphasis on the practical application of research, as well as developing software engineering methods for building complex systems.