Development of a Component-based GIS using GRASS

Xueming Wu, Shifeng Zhang and Steve Goddard

Department of computer Science and Engineering
University of Nebraska-Lincoln
Lincoln, Nebraska 68588-0115, U.S.A
{xwu, shzhang, goddard}@cse.unl.edu

1. Introduction

Due to the driving forces of the Internet and network communication technology, the paradigm of Geographic Information Systems (GIS) is shifting [16, 19]. Moreover, with the advances in computer technology and GIS science, distributed GIS become practical and widely acceptable. Traditional GIS plays an extremely valuable role in GIS applications by providing a wide variety of tools to handle geo-referenced data. However traditional GIS cannot be applied directly in distributed and heterogeneous computing environments due to their closed and centralized architectures.

To hide the heterogeneity and to accommodate distributed environments, open, component-based GIS are replacing the old traditional monolithic GIS [2, 6, 9]. Under the open, component-based architecture, distributed data and functionality can be integrated and cooperate with each other and at the same time it minimize the risk of developing new monoliths. Furthermore, by applying component-based software development (CSBD) approaches, components can be reused and composed in many applications. Component technologies, such as the Common Object Request Broker Architecture (CORBA), Distributed Component Object Model (DCOM), and Java Remote Method Invocation (RMI), are playing major roles in the construction of open, component-based architectures.

While many component-based GISs have been developed from scratch, there is a need to develop a component-based GIS based on existing code rather than development from scratch. For instance, constructing systems from GIS commercial off-the-shelf (COTS) products have been conducted [20]. The advantages of using previously-existing systems are numerous including tested reliability, approved features, and an opportunity for expanding system capabilities [20]. So, building a component-based GIS from an existing traditional GIS is a research problem that we are trying to address.

This paper introduces an approach to transform a traditional GIS, GRASS, into a component-based GIS under CORBA environment. Section 2 first introduces component-based software development and CORBA. In Section 3 we describe the layered architecture and related methods and technologies used to transform GRASS into a component-based GIS. A case study that integrates the GIS components is presented and evaluated in Section 4. Section 5 presents the conclusions and our contributions.

2. Background

This section presents related background knowledge. Section 2.1 introduces component-based software development. CORBA is briefly introduced in Section 2.2.

2.1 Component-based Software Development

A software component is a self-contained unit which can be independently deployed and is subject to composition by third parties [18]. Software components adopt features of object-oriented programming including encapsulation, polymorphism, inheritance, object-binding, and object relationships such as specialization, collaboration and composition [17].

CSBD allows systems to be developed from a number of existing software components with exposed interfaces and hidden implementations. Thus, a system can be developed by selecting, reconfiguring and assembling encapsulated, reusable, interoperable, pre-testing software components [1]. The major benefits of component-based software development include shortened development cycles, increasing productivity, and higher quality systems [7].

Currently there are three major component technologies used in the development of component-based applications. They are CORBA specification developed by the Object Management Group (OMG), DCOM developed by Microsoft Corporation and Java RMI developed by Sun Microsystem Inc. These component technologies have been widely adopted by the GIS community. For example, OpenGIS Consortium (OGC) issued the Simple Feature (SF) [13] and Grid/Coverage (GC) [12] geospatial data implementation specifications for CORBA and DCOM, which can be used as the standard geospatial data representations in CORBA, DCOM, or Java RMI environments. Environmental Systems Research Institute's (ESRI) ArcGIS 8.3 and higher were developed for the DCOM environment [4].

2.2 Introduction to CORBA

CORBA is a well-accepted, mainstream component technology. It targets the problems associated with heterogeneity in distributed computing environments. Such heterogeneity is common because platform-dependent computing technology changes over time, e.g., the operating systems and the network technology. CORBA, as a platform-independent computing model and abstraction, can not only hide the heterogeneity between different platforms, but also hide the complexity in the low-level network communication. It provides a standardized interface model and object framework for solving network computing problems in a distributed heterogeneous environment.

The CORBA architecture consists of four main parts: Object Request Broker (ORB), Common Object Services, Common Facilities, and Application Objects. Essential to CORBA's architecture is the ORB. The ORB is responsible for distributing object calls between clients and servers. The object calls can be either static or dynamic. Figure 1 illustrates the role of the ORB. The protocol for client/server interaction is defined through a single implementation language-independent specification, Interface Definition Language (IDL). The purpose of the IDL is to allow the definition of the object interfaces to be independent of any particular programming languages and provide operating-system-independent interfaces to the services and components which reside on a CORBA bus.

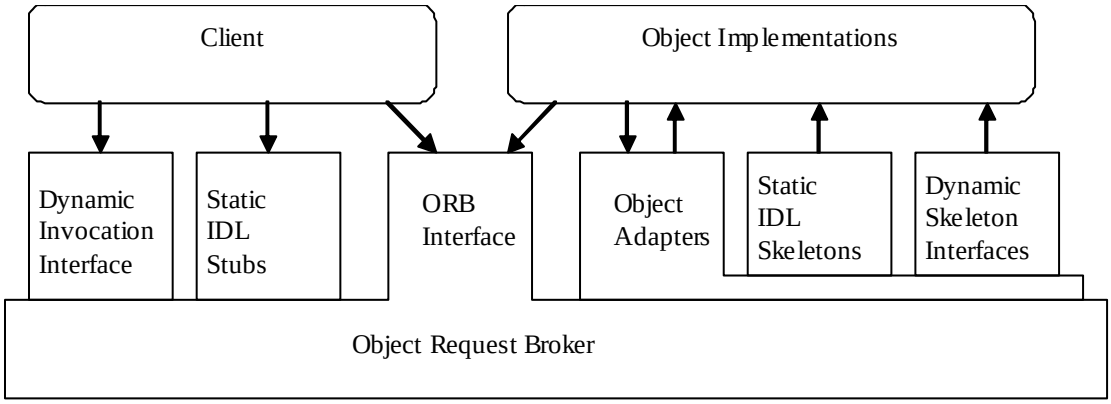

Figure 1: The structure of object request interfaces [14].

Many of today's GIS applications in distributed environments consist of components from different knowledge-domains which are both technically and semantically heterogeneous. Such heterogeneity can be overcome by using CORBA.

3. Architecture and Methods

We propose a layered architecture and methods with which a component-based distributed GIS can be built using a traditional GIS such as GRASS. The transformation of GRASS from a traditional GIS into a component-based distributed GIS is based on the proposed layered architecture which is introduced in Section 3.1, and achieved by encapsulating the GRASS commands in a shared C library called GRASSLib which is described in Section 3.2. Section 3.3 describes elements of the component-based GIS. Some implementation related issues are presented in Section 3.4.

3.1 Architecture

Figure 2 represents the general layered architecture of a component-based GIS. Each layer in this architecture is independent of its underlying layer given that the interface of the underlying layer does not change. For example, following the design rules of layered architecture [18], objects in the layer hosting a *component-based GIS Server* access the *Traditional GIS Kernel* layer exclusively via the *GIS Library* layer. Therefore changes to the *Traditional GIS Kernel* will not require an adaptation of the classes or tools built within the *component-based GIS Server* layer.

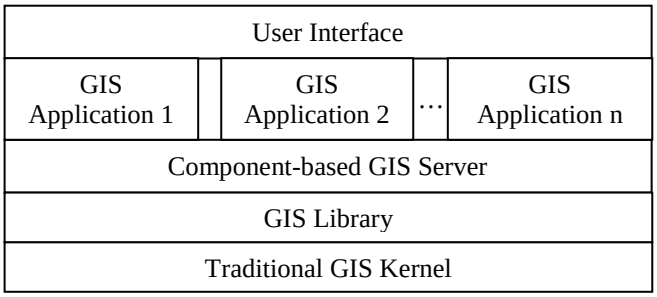

Figure 2: General architecture.

A *Traditional GIS Kernel* layer provides basic spatial data management and processing functions; however it cannot be readily used in a distributed environment. For example, GRASS is a command-oriented GIS which cannot directly accommodate distributed computing environments. The *GIS Library* layer provides sophisticated geo-processing services based on the underlying *Traditional GIS Kernel* layer. The *GIS Library* essentially wraps the traditional GIS commands or tools as a library with open APIs. This approach makes a traditional command-oriented GIS available in a distributed environment. A *Component-based GIS Server* is built on top of the *GIS Library*. This server provides domain-related geo-processing and mapping services to the layer built on top of it. Through this server, the complexity of the traditional GIS is hidden. Different GIS can be integrated in the system. For example either ESRI ArcGIS or GRASS can be used as the *Traditional GIS Kernel*. This server can be implemented as several separate components to gain flexibility and efficiency. GIS Applications are built on top of the *Component-based GIS Server*. Each application can be implemented as a set of separate components. And those parts of the application related to spatial data management and processing are implemented by the *Component-based GIS Server*. Therefore GIS applications are independent of the underlying traditional GIS.

Figure 3 presents an instance of the general layered architecture, which transforms GRASS from a traditional GIS into a component-based GIS. In Figure 3, GRASS serves as the base of the layered architecture by providing the basic spatial data management and spatial analysis functions. The GRASS GIS Library is the primary C programming library provided by GRASS. It is the kernel of GRASS. GRASSLib is a shared C library developed by wrapping GRASS commands or related tools with open APIs. More details of GRASSLib are described in Section 3.2. The Component-based GRASS GIS Server is built by utilizing CORBA technology and GRASSLib. Each component of the server provides GIS Applications with specific geo-processing and mapping services which are accomplished using GRASSLib. Through the transformation, the component-based GRASS GIS Server can accommodate distributed computing environments. Moreover, the components in the server can be integrated in different applications.

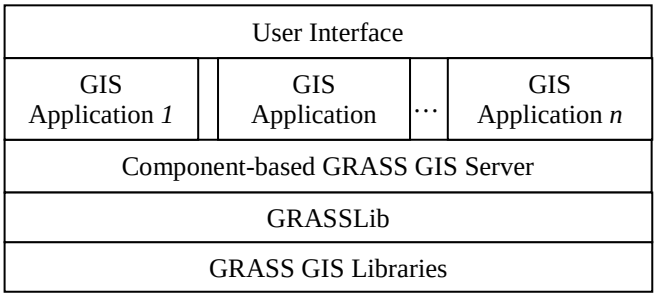

Figure 3: The architecture of component-based GRASS.

3.2 GRASSLib

GRASSLib plays a crucial role in transforming GRASS from a traditional GIS into a component-based GIS. It is developed by wrapping core functions and related tools of GRASS into a shared library, which can be linked into components. It converts GRASS, a traditional GIS, from its original command-oriented style into an executable component. Therefore GRASS can be integrated into component-based applications. Thus the core spatial data management and processing functions of GRASS can be provided to other components through an object-oriented method in a distributed environment such as CORBA, DCOM, and JAVA RMI.

Each command of GRASS was implemented by several C programs calling the functions provided by GRASS GIS Libraries. To change a GRASS command into a function of GRASSLib, several important modifications have been made to the original programs supporting the command. The key modifications include:

- The main() subroutines are changed to ordinary functions that take all the command line parameters of the GRASS command as parameters passed in by a client. An integer will be returned by the function to indicate whether the execution is successful. Figure 4 shows the declaration of the function converted from GRASS command s.surf.rst.

```
DLL_EXPORT int spline_i(char* arg_input, char* arg_maskmap, char* arg_elev, char*
arg_devi, char* arg_slope, char* arg_aspect, char* arg_pcurv, char* arg_tcurv, char*
arg_mcurv, char* arg_treefile, char* arg_overfile, int arg_deriv,  int arg_dtens, double
arg_dmin, double arg_fi, int arg_KMAX, int arg_npmin, double arg_zmult, int arg_elattr,
double arg_theta, double arg_scalex, double arg_rsm, int arg_smattr, int mapoption );
```

Figure 4: Declaration of a function in GRASSLib.

- Eliminate the use of global variables and static variables. Some of these variables have been changed into parameters passed to related functions. The others will be reset to initial values at the end of each execution.
- Free memory occupied by the programs at the end of each execution to reduce memory leaks. Figure 5 shows an example of memory de-allocation and resetting of the global variables in Spline interpolation function. Memory management is important to transform GRASS commands into shared library. From our experience, it is time-consuming to detect and eliminate the potential memory leak in the GRASSLib.

```
#define CLEAN_BEFORE_RET \
{\
   if (az) G_free_vector(az); az = NULL; \
   if (adx) G_free_vector(adx); adx = NULL; \
   if (ady) G_free_vector(ady); ady = NULL; \
   if (adxx) G_free_vector(adxx); adxx = NULL; \
   if (adyy) G_free_vector(adyy); adyy = NULL; \
   if (adxy) G_free_vector(adxy); adxy = NULL; \
   if (functions) free(functions); functions = NULL;   \
   if (info->root->data->points){ free(info->root->data->points); printf("free data->points\n"); };   \
   if (info->root->data) { free(info->root->data); info->root->data = NULL; printf("free data \n");};  \
   if (info->root) free(info->root); info->root = NULL;        \
   if (info) free(info); info = NULL;                          \
   if (zero_array_cell) free(zero_array_cell); zero_array_cell = NULL; \
   if (bitmask) BM_destroy(bitmask);  bitmask = NULL; \
   if (fddevi) fclose(fddevi); fddevi = NULL;          \
   if (elev) fclose(Tmp_fd_z); Tmp_fd_z = NULL;    \
   if (slope) fclose(Tmp_fd_dx); Tmp_fd_dx = NULL;  \
   if (aspect) fclose(Tmp_fd_dy); Tmp_fd_dy = NULL;  \
   if (pcurv) fclose(Tmp_fd_xx); Tmp_fd_xx = NULL;    \
   if (tcurv) fclose(Tmp_fd_yy); Tmp_fd_yy = NULL;     \
   if (mcurv) fclose(Tmp_fd_xy); Tmp_fd_xy = NULL;      \
   if (Tmp_file_z) free(Tmp_file_z); Tmp_file_z = NULL;      \
   if (Tmp_file_dx) free(Tmp_file_dx); Tmp_file_dx = NULL;      \
   if (Tmp_file_dy) free(Tmp_file_dy); Tmp_file_dy = NULL;      \
   if (Tmp_file_xx) free(Tmp_file_xx); Tmp_file_xx = NULL;      \
   if (Tmp_file_yy) free(Tmp_file_yy); Tmp_file_yy = NULL;      \
   if (Tmp_file_xy) free(Tmp_file_xy); Tmp_file_xy = NULL;      \
}\
```

Figure 5: Example code of memory de-allocation.

Currently there are seventeen GRASS commands from spatial interpolation to geospatial output that can be called directly from the GRASSLib. The API of the GRASSLib can be found on the project website [11]. More commands can be added to GRASSLib and be exposed to other components.

3.3 Components of the Component-based GRASS GIS Server

The Component-based GRASS GIS Server is built on top of GRASSLib. It provides geo-processing and mapping services to the layer built on top of it. Through this server, the complexity of the traditional GIS is transparent to GIS applications. As illustrated in Figure 6, currently the component-based GRASS GIS server is comprised of three components. The SpatialSupportObject provides some basic spatial operations such as interpolation, reclassification, and spatial data conversion. The SpatialDataObject is responsible for listing available spatial data and metadata of the spatial data in the datasets and retrieving spatial data from the dataset. The SpatialInfoObject supplies clients with mapping tools. These tools can be used to transform spatial data into a pdf, png or gif file.

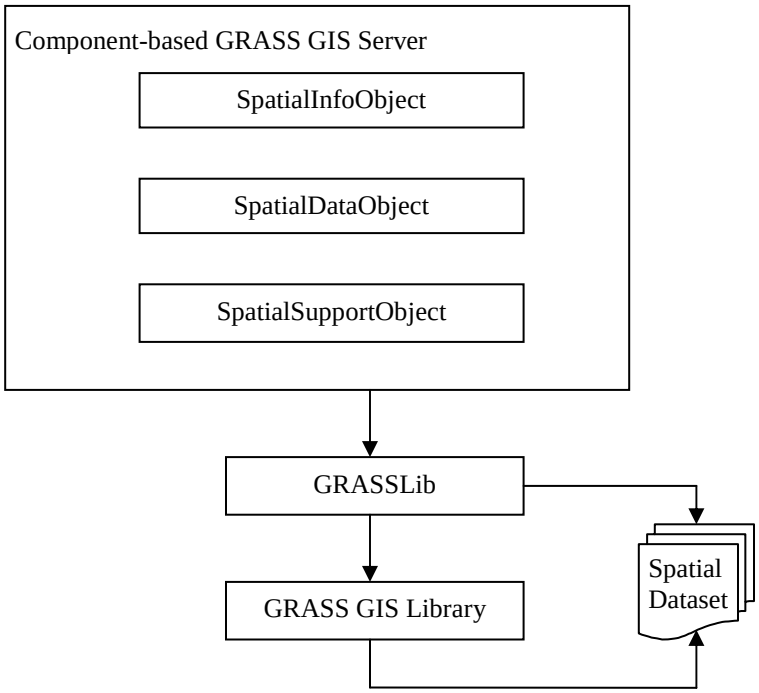

Figure 6: Components of GRASS GIS Servers.

These components are implemented as CORBA objects. The interface of each component is defined using IDL. Figure 7 presents the IDL definition of the SpatialDataObject while Figure 8 gives its C++ class definition in OmniORB environment. Through IDL, the interface of each component is exposed to the client application that requests the services while the implementation of the component is hidden. This is one of the benefits provided by component technologies.

```
interface SpatialDataObject {
        BinaryLayer getSite(in BaseMap area, in string siteName, in DataFormat
destination) raises (SpatialOperationException, InvalidParameterException,
MapFormatNotAvailableException);
        BinaryLayer getVector(in BaseMap area, in OverlayType type, in
MapResolution resolution, in DataFormat destination) raises (SpatialOperationException,
InvalidParameterException, MapFormatNotAvailableException);
        BinaryLayer getRaster(in BaseMap area, in OverlayType type, in MapResolution
resolution, in DataFormat destination) raises (SpatialOperationException,
InvalidParameterException, MapFormatNotAvailableException);
        LayerHeaderSeq listSite(in BaseMap area) raises (SpatialOperationException,
InvalidParameterException);
        LayerHeaderSeq listVector(in BaseMap area) raises (SpatialOperationException,
InvalidParameterException);
        LayerHeaderSeq listRaster(in BaseMap area) raises (SpatialOperationException,
InvalidParameterException);
    };
```

Figure 7: IDL Interface of SpatialDataObject.

```
class SpatialDataObject_i: public POA_SpatialLayer::SpatialDataObject,
        public PortableServer::RefCountServantBase {
    private:
        PortableServer::POA* mypoa;
        char config_file[512];
        omni_mutex grass_mutex;
    public:
        SpatialDataObject_i(const char* configfile, PortableServer::POA *p);
        virtual ~SpatialDataObject_i();
        BinaryLayer* getSite(const BaseMap& area, const char* siteName, const DataFormat&
destination) throw  (SpatialOperationException, InvalidParameterException,
MapFormatNotAvailableException);
        BinaryLayer* getVector(const BaseMap& area, OverlayType type, const MapResolution&
resolution, const DataFormat& destination) throw  (SpatialOperationException,
InvalidParameterException, MapFormatNotAvailableException);
        BinaryLayer* getRaster(const BaseMap& area, OverlayType type, const MapResolution&
resolution, const DataFormat& destination) throw  (SpatialOperationException,
InvalidParameterException, MapFormatNotAvailableException);
        LayerHeaderSeq* listSite(const BaseMap& area)
            throw (SpatialOperationException, InvalidParameterException);
        LayerHeaderSeq* listVector(const BaseMap& area)
            throw (SpatialOperationException, InvalidParameterException);
        LayerHeaderSeq* listRaster(const BaseMap& area)
            throw (SpatialOperationException, InvalidParameterException);
        void destroy();
};
```

Figure 8: Class definition of SpatialDataObject.

3.4 Implementation Issues

GRASS is a complex system. Only wrapping the GRASS commands into a shared library is not enough to apply them to a component environment. This section introduces the related technologies used to transform GRASS into a component-based GIS server. Section 3.4.1 presents the management of GRASS mapsets in component environment. Section 3.4.2 covers the mechanism used in projection transformation. The data formats supported by the component-based GIS server are described in Section 3.4.3. Section 3.4.4 explains the synchronization issue. The component communication mechanism is presented in Section 3.4.5.

3.4.1 Management of Mapsets

All the GRASS programs and tools must be executed in a mapset. Settings of a mapset such as projection, region, and mask will affect the result of the execution. When a service provided by the Component-based GRASS GIS Server is requested, a thread is spawned by the server to handle the request. The thread needs a mapset and that thread will use the mapset exclusively. Our approach to assign and release mapsets is based to the approach used in GRASSLINKS [8].

Besides the PERMANENT mapset, a number of mapsets are created in each LOCATION. These mapsets serve as working mapsets for any thread spawned by the Component-based GRASS GIS Server. We use a lock mechanism to prevent a mapset from being used by more than one thread at the same time. An empty file called UNLOCK is generated while a mapset is created. Right before the calling of a function of GRASSLib in the thread, the server will check the mapsets in

the current LOCATION cyclically. The first mapset found with a UNLOCK file will be assigned to the thread and becomes the current working mapset for the thread. Once a mapset is assigned to a thread, file UNLOCK will be renamed to LOCK to indicate that the mapset is in use. Before renaming UNLOCK to LOCK, the server will check if a file called LOCK already exists. If file LOCK exists, that means the mapset is in use. The server will continue checking other mapsets. After the execution of the thread, that file will be renamed to UNLOCK again. Then that mapset can be available to any other thread.

Once a mapset is assigned to a thread, the attributes of the mapset such as region and mask will be set up according to parameters passed to the function by the client. The settings will impact the results of the invoked method.

3.4.2 Location Management and Projection transformation

Projection transformation is an important and necessary functionality of a GIS. The Component-based GRASS GIS Server also supports this functionality. The server uses two approaches to conduct projection transformation. These two approaches are an explicit method and an implicit method. The explicit method transforms the coordinates from a given projection to a required projection by directly processing the coordinates based on the API provided by the GRASS GIS library. The explicit method is applied to the client input parameters that required projection transformation. These parameters usually only contain a few pairs of coordinates.

The implicit method performs projection transformation using the setting of GRASS LOCATION. The LOCATION is one part of the hierarchy of the GRASS database structure. The LOCATION is actually a directory in a UNIX file system. And mapsets are sub-directories of a certain LOCATION in a UNIX file system. In GRASS each LOCATION has a certain projection setting. All the mapsets that belong to the same LOCATION share the same projection setting. The implicit method conducts the transformation by setting a mapset belonging to a LOCATION with required projection as the current working mapset. Before the server assigns an available mapset to a thread, the server first determines which LOCATION to choose as the current LOCATION. Then the server will assign an available mapset of that chosen LOCATION to the requested thread.

The implicit approach is applied to the output of the server, which may be a raster or a vector. This approach has no computing overhead compared to the explicit approach. Therefore the Component-based GRASS GIS Server can gain performance. However, performance comes at a price of disk space and possibility of data inconsistencies. The server creates one LOCATION for each projection that it supports. And a copy of the base maps in the defined projection will be stored in each LOCATION.

3.4.3 Data Formats

The Component-based GRASS GIS Server adopts the GRASS spatial data model and uses the GRASS data format as its native format. However, to prevent the Component-based GRASS GIS Server from being closed and monolithic, the input and output spatial data the server uses the standard ASCII-based vector and raster formats that are supported in many GIS software. The Component-based GRASS GIS Server also provides functions to perform data conversion between the GRASS data format and other formats such as ESRI shape files, Acrobat pdf, jpeg and gif files. Since pdf, jpeg and gif files can be viewed through ordinary web browsers, the

Component-based GRASS GIS Server can be used in a web GIS application which is developed and deployed in a distributed environment such as CORBA, DCOM, and JAVA RMI.

3.4.4 Synchronisation

GRASS is a traditional command-oriented style GIS. There are some shared resources used in GRASS such as environment variables and the .grassrc5 file. Thus we have to provide a method for protecting the shared resources used in GRASS when we transform GRASS into a component-based GIS that support multithreads. We use class omni_mutex to solve this problem. A variable of type omni_mutex is defined in the implementation of each component (shown in Figure 8). A class omni_mutex provides two operations, lock() and unlock(). Wherever a GRASSLib function is called, the function is placed between a pair of lock() and unlock() calls. Hence the shared resources are protected in a consistent manner throughout the system.

3.4.5 Communication Mechanism

The inter-object and intra-object communication mechanism is provided by a CORBA ORB. The ORB itself is an object. When a client object requests a method on a server object, it goes to the ORB. The ORB then invokes the method on behalf of the client. The ORB takes care of all the tedious tasks of locating the server object, establishing a connection, invoking the method, getting the result, and closing the communication session. The CORBA Naming Service is used to locate objects by name.

4. An Example: National Agriculture Decision Support System

The National Agricultural Decision Support System (NADSS) is a web-based Spatial Decision Support System (SDSS) [5]. It provides geospatial 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 (e.g., the impact of a natural hazard). NADSS consists of spatial analytical models to use climatic data to generate drought maps based on indices such as the Standard Precipitation Index (SPI) [10] and the Palmer Drought Severity Index (PDSI) [15].

4.1 Architecture

Figure 9 shows the architecture of NADSS, which is built on top of the Component-based GRASS GIS Server. NADSS is comprised of a set of independent components that can be deployed on several computers. NADSS utilizes CORBA for GIS infrastructure and Enterprise Java Beans (EJB) for application distribution [3]. Therefore NADSS can accommodate a heterogeneous distributed computing environment. By assembling the Component-based GRASS GIS Server in the system, the complexity of traditional GRASS is transparent to application components and interface components.

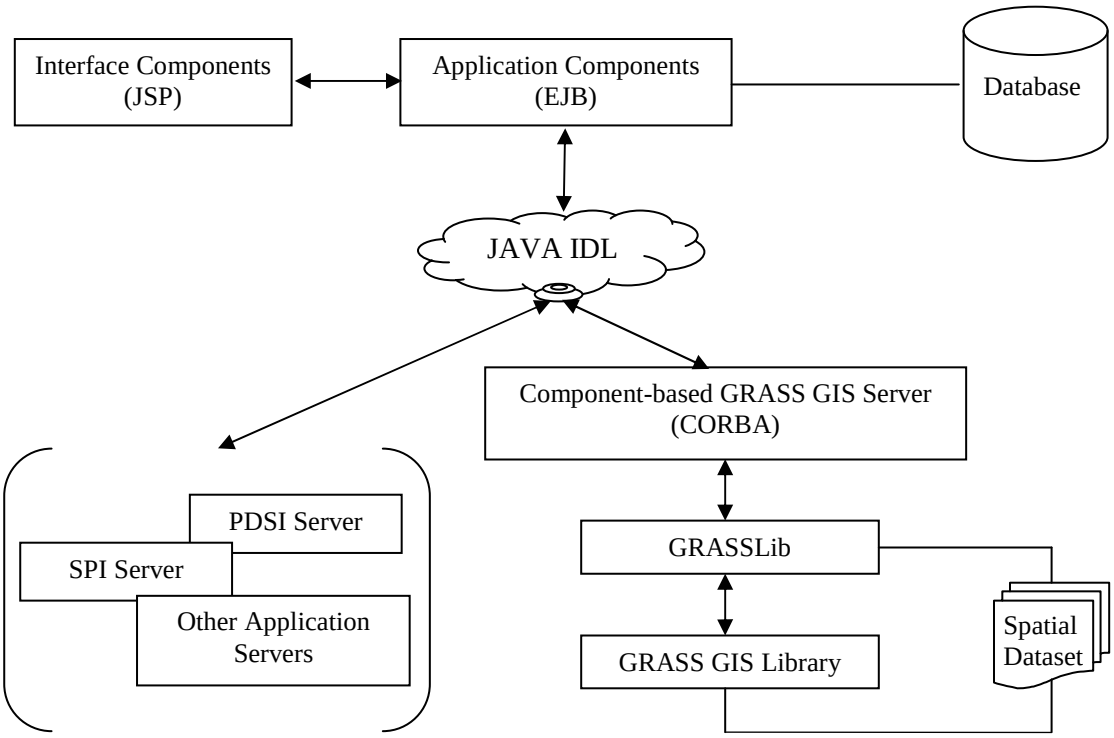

Figure 9: Architecture of NADSS.

Figure 10 demonstrates the generation a PDSI map of a selected area in Nebraska in NADSS. In Step 1, the client sent a request to the PDSI Server through the ORB. The PDSI Server calculated the PDSI for the selected area and returned the PDSI information to the client via the ORB. Then in Step 2, the client passed the PDSI information and a request for a map to the Component-based GRASS GIS Server via the ORB. The Component-based GRASS GIS Server conducted an interpolation using the PDSI information, generated a PDSI map, and converted the map to the required format such as pdf, png or gif. Then the server returned the generated map to the client through the ORB.

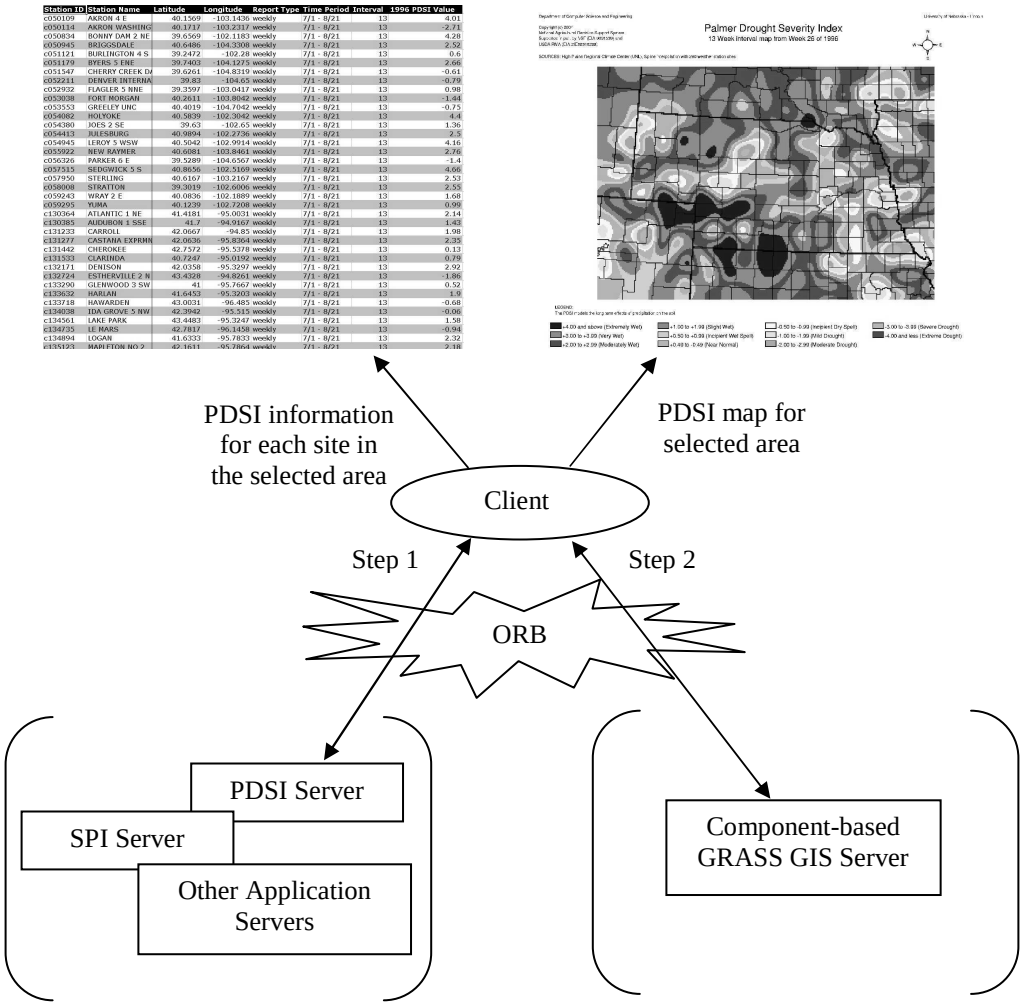

Figure 10: Generation of a PDSI map.

4.2 Performance Evaluation

To transform a traditional GIS in a distributed environment can extend the usage of the traditional GIS. However, the transformation may impact the performance of the system. We evaluate the performance of our implementation and present the results.

To evaluate the performance of the component-based GRASS GIS server implemented in NADSS, we compared interpolation functions from the server and a C program which called the interpolation command of GRASS (s.surf.rst was used in the evaluation) through a system call. These two interpolations used the same input data, same interpolation method and parameters. The component-based GRASS GIS server first retrieved input data, then called the interpolation function of GRASSLib, and called the raster conversion function of GRASSLib to convert the interpolation result into ASCII-raster format. Finally the ASCII raster was sent to the client using the CORBA IIOP protocol. The C program first called the interpolation command of GRASS, which read the input data from a GRASS site file, then called the raster conversion command of GRASS (r.in.ascii was used) to convert the interpolation result into ASCII format.

This evaluation was designed to estimate the computation overhead caused by wrapping the GRASS commands with component technology. However, there exists communication overhead between the client and the component-based GRASS GIS server. To minimize the impact of transferring raster data from the component-based GRASS GIS server to client, the server and the client were deployed on the same computer in the evaluation.

We tested the average processing time in seconds for four different spatial resolutions (3000m [low resolution], 1000m, 500m, 200m [high resolution]). The different resolutions affect the interpolation result size and consequently the raster data size (the greater the resolution, the larger the raster data size).

Table 1 presents the average call times and standard error of call times from the test samples of the server and the C program at each resolution. The average call times are plotted in Figure 11 to provide a visual comparison of their performance. From the results, we found that the component-base GRASS GIS server was a little slower than the C program, which called GRASS commands directly. The performance impact comes from the communication overhead of the CORBA components. Nevertheless, the performance impact is acceptable and the component-based GRASS GIS server can accommodate distributed computing environments.

	Component-based GRASS GIS server		**C program (GRASS command)**	
	mean	standard error	mean	standard error
3000x3000m	0.885201	0.019641	0.855673	0.002863
1000x1000m	6.920380	0.117729	6.464280	0.047110
500x500m	27.098860	0.189585	25.248723	0.090016
200x200m	167.303000	0.665325	158.480230	2.793640

Table 1: The mean call time in seconds and the standard error of call times from the test samples for the Component-based GRASS GIS server and GRASS commands.

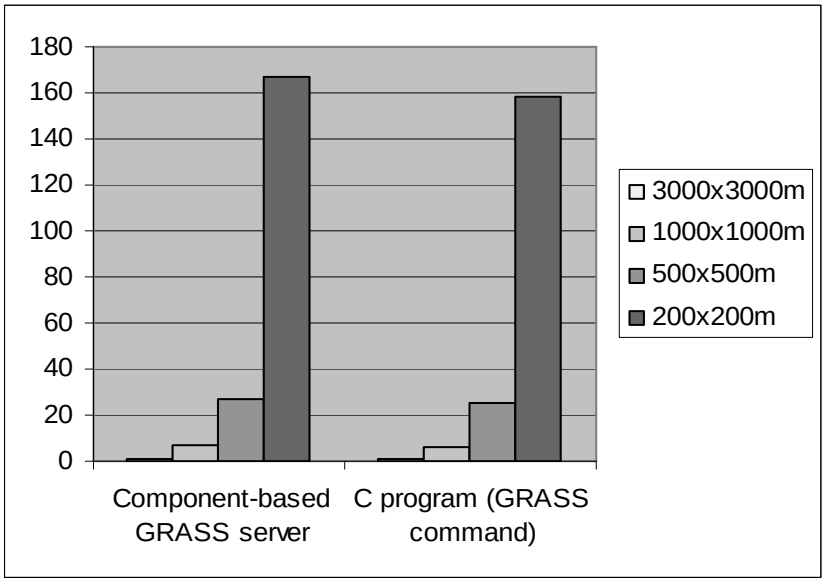

Figure 11: Performance evaluation.

5. Conclusion

An approach to transform GRASS, a traditional GIS, into a component-based GIS server has been presented and an example of integrating the component-based GIS server in a SDSS is given. We showed that this approach is useful and practical by applying component technology to a traditional GIS to achieve accommodation of distributed computing environments.

First, we introduced a layered architecture that uses component technology to construct a component-based GIS server from a traditional command-oriented GIS such as GRASS. A layer in the layered architecture is independent of its underlying layer, given that the interface of the underlying layer does not change.

Second, we presented our work on GRASSLib, which wraps core functions and tools of GRASS with component technology into a shared library. It converts GRASS from its original command-oriented style into an executable component which can be linked to other components through an object-oriented method in a distributed environment.

Third, we described the implementation of a component-based GRASS GIS server built on top of GRASSLib. Each component of the server is a CORBA object that provides specific geoprocessing and mapping service to GIS application components. Some methods related to the management of GRASS mapsets and locations were also covered in this paper.

We used NADSS, a SDSS, as a case study to demonstrate the application of the component-based GRASS GIS server. We also presented a performance evaluation of the example server.

Traditional GIS software, like GRASS, are command-oriented, and normally unsuitable for use in a component-based distributed computing environment. This limitation has prevented their broad usage in today's distributed GIS applications. The transformation approach can be applied to similar traditional GIS whose core functions and tools can be wrapped using shared libraries.

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

References

[1] Barroca, L., Hall, J., and Hall, P., edtors. (2000). Software Architectures: Advances and Applications. Springer-Verlag, London, UK.

[2] Coddington, P.D., Hawick, K.A., Kerry, K.E., Mathew, J.A., Silis, D.L., Webb, P.J., Whitbread, C.G., Irving, M.W., Grigg, R., and Jana, KT. (1998). Implementation of a Geospatial Imagery Digital Library using Java and CORBA. Proc of Technologies of Object-Oriented Languages and Systems (TOOLS) Asia'98, Beijing, China, pp. 280-290.

[3] Cottingham, I.J., Goddard, S., Zhang, S., Wu, X., Lu, K., Rultedge, A., and Waltman, W. (2004). Demonstration of the National Agriculture Decision Support Systems. Proc of 2004 national conference for digital government research, Seattle, WA. pp. 303-305.

[4] Environmental Systems Research Institute (ESRI), ArcGIS. http://www.esri.com. (accessed May 12, 2004).

[5] 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.
[6] Goddard, S., Harms, S., Reichenbach, S., Tadesse T., and Waltman, W (2003). Geospatial Decision Support for Drought Risk Management. Communication of the ACM, Vol 46, No. 1, pp. 35-37.
[7] Haines, C.G., Carney, D., and Foreman, J. (1997). Component-Based Software Development/ COTS Integration. http://www.sei.cmu.edu/str/descriptions/cbsd_body.html (accessed June 2, 2004).
[8] 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.
[9] Jacobsen, H. A. and Voisard, A. (1998). CORBA-Based Interoperable Geographic Information Systems. Proc. of Euro Parallel and Distributed Systems Conference, Vienna.
[10] McKee, T.B, Doesken, N.J, and Kleist, J. (1993). The Relationship of Drought Frequency and Duration to Time Scales. Proc of 8th Conference on Applied Climatology. pp. 179-184.
[11] NADSS GRASSLIB website: http://nadss.unl.edu/grasslib (accessed Dec 26, 2003)
[12] OpenGIS Consortium Inc. (2001). OpenGIS Grid Coverage Implementation Specification, OGC Project Document Number 01-004.
[13] OpenGIS Consortium Inc. (1998). OpenGIS Simple Features Implementation Specification for CORBA, OGC Project Document Number 99-054
[14] Object Management Group (OMG). (1998). Them Common Object Request Broker: Architecture and Specification, 2.2 ed. Famingham, Massachusetts: OMG.
[15] Palmer, W.C. (1965). Meteorological Drought. Research Paper No. 45, U.S. Department of Commerce Weather Bureau, Washington D.C.
[16] Preston, M., Clayton, P., and Wells, G. (2003). Dynamic Run-time Application Development Using CORBA Objects and XML in The Field of Distributed GIS. International Journal of Geographical Information Science, Vol. 17, No. 4, pp321-341.
[17] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy F., and Lorensen, W. (1991). Object-oriented Modeling and Design, Englewood Cliffs, New Jersey, Prentice-Hall, Inc.
[18] Szyperski, C. (1998). Component Software: Beyond Object-Oriented Programming. Harlow, Addison-Wesley.
[19] Tsou, M.H. and Buttenfield, B.P. (2002). A Dynamic Architecture for Distributed Geographic Information Services. Transactions in GIS. 6(4), pp. 335-381.
[20] Tu, S., Xu, L., Abdelguerfi, M., and Ratcliff, J. J. (2002). Achieving Interoperability for Integration of Heterogeneous COTS Geographic Information Systems. Proc. of the 10th ACM International Symposium on Advances in Geographic Information System, McLean, VA, pp 162-167, November 2002.