An adaptive package architecture for corporate GIS

Andrés Pazos, José Poveda, Michael Gould
Department of Information Systems (LSI), Universitat Jaume I
E-12071 Castellón, Spain
apazos@iac.es
{albalade, gould}@uji.es

SUMMARY

We introduce a flexible software architecture that overcomes some of the common problems found in corporate GIS environments, especially in the public sector. This three-tier architecture defines a set of software packages that are composed at the server side in response to the needs of each user. The kernel and n external packages are downloaded and configured at run-time, with a middleware layer handling authentication and versioning. To demonstrate the proposed architecture, we present a practical development use case that facilitates the composition of separate free software packages in the GIS domain.

KEYWORDS: *Distributed corporate GIS, network-based GIS architecture, component-based system*

INTRODUCTION

Corporate GIS are multi-participant, multi-user computing environments, and while they are not necessarily complex (often just a single server and simple client applications) they do tend to be difficult to support due to the large number of users, working in multiple, geographically dispersed offices. This complicates practical tasks such as data sharing among departments, system installations, upgrades and routine maintenance. Each of potentially thousands of users may have specific requirements based on their capacity to use the GIS, and requirements regarding hardware and software configurations: one size does not fit all. Status quo has been that system administrators are forced to install custom upgrades and extensions one workstation at a time, either that or offer all users the same monolithic package to be downloaded remotely. Until recently GIS software was largely monolithic (see figure 1); even though experience shows that many users exercise only a fraction of the functionality offered by a full-featured GIS, system administrators were required to install and support a full, often costly, commercial license for each and every user. The leading GIS vendors offer so-called extensions to a base GIS product, but what happens when that base already offers excessive, or just not the right mix of, functionality for a particular user? In search of just the right set of functions for each user, two primary options have become available: 1) end user development (EUD) which assumes the user has software development skills (Morch, 2004) to be able to build an application from JavaBeans or ActiveX components, or 2) server-side configuration of software modules, here termed *packages* after the terminology common in the Java, Perl and Python communities. We opt for this second option, which for the reader familiar with ESRI terminology (and purely for comparison) would fall somewhere between a monolithic ArcView and the manual programming of smaller ArcObjects.

In organizations with an elevated number of users, and especially in the public sector which is extremely cost-conscious, an extensible and easily-customizable GIS application solution is needed. This is possible following two levels of application flexibility: at the organizational level the system presents scalability that allows reuse of existing software, adapted to evolving needs; at the user level, he or she should determine the customization needed, periodically installing or updating only the parts of the application needed to perform his or her work (Morch, 1997).

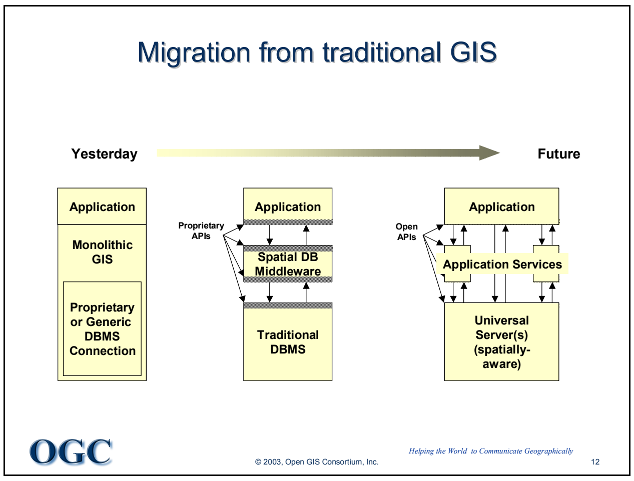

Migration from traditional GIS
Yesterday
Future
Application
Monolithic GIS
Proprietary or Generic DBMS Connection
Proprietary APIs
Application
Spatial DB Middleware
Traditional DBMS
Open APIs
Application
Application Services
Universal Server(s) (spatially-aware)
OGC
Helping the World to Communicate Geographically
© 2003, Open GIS Consortium, Inc.
12

Figure 1. The migration from monolithic to service-based GIS architectures, according to OGC. Note that OGC defines interfaces (here termed Open APIs) yet does not define how to implement them.

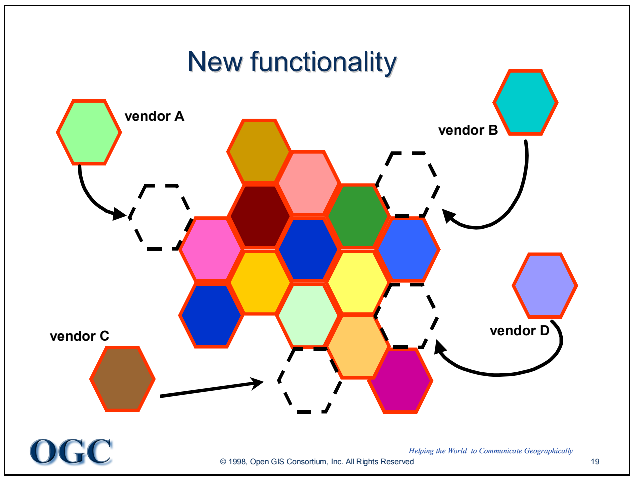

New functionality
vendor A
vendor B
vendor C
vendor D
OGC
Helping the World to Communicate Geographically
© 1998, Open GIS Consortium, Inc. All Rights Reserved
19

Figure 2. The OGC vision of interchangeable COTS components; OGC does not define how this process should be implemented.

The component-based software development (CBSD) paradigm (Brown, 2000) is helping to make this extensibility and flexibility a reality. In the CBSD approach new code development is minimized and system upgrades become the task of replacement of well-bounded functional units of the system. This replacement concept is not unlike that suggested by OGC (see figure 2) however we remind the reader that OGC defines only interface specifications, not architectures or implementation alternatives, which is what we describe here. In our proposed architecture the components are grouped in packages, defined as a functional, replaceable set of services. This allows acquisition of packages developed from third-party developers (COTS or free software) and adapting them to the

CBSD process. These applications may be composed of a light kernel, which implements the basic functionality, and a set of independent packages each offering extended functionality. Each package implements a specific part of the GIS and is connected to the kernel through published interfaces (not necessarily OGC) in order to compose the final application desired by the user. Note that the composition or integration does not happen automatically; the system administrator still must build wrappers or connectors, however this is done for each new package to be included for potential use by all users, not something that is programmed by or for each user. Programming is centralized and application configuration decentralized.

Thanks to the ubiquitous Internet infrastructure it is now routine to download and install custom software rather than relying on CDs or other hard media for full installations. However, conventional client/server architectures do not cope well with access control and distribution requirements, forcing us to move to a 3-tier software architecture. A central server stores and dispatches the different pieces (packages) that may be combined to form custom final applications, while the middleware provides a natural layer where to locate the adaptive, context dependent behavior (McKinley, 2004). The client interface is through a specific HTTP browser that connects to the middleware services. The system administrator takes charge of controlling the server, making the necessary changes in order to adapt the packages to the CBSD process. The system administrator may also monitor download and 'real' system usage among the various users, and make adjustments, load balancing for example, accordingly.

The case study that motivated this work is taken from our experience with public administration, specifically the Infrastructure and Transport Department of a regional government authority. This organization supports more than four hundred computer users, many of whom utilize GIS only on a sporadic basis. They commonly use proprietary desktop applications with high annual license payments. According to internal surveys, many of these users require only a specific subset of complete GIS functionality. Moreover, the users are distributed geographically, so it is important to centralize remote software installation and maintenance in order to save time and money. Finally, these users generally prefer to have the GIS application loaded locally, rather than depend upon full-time web services access.

The architecture proposed here is valid for commercial as well as free software, but additional monetary benefit naturally accrues from the configuration and composition of GIS clients based on free software components (Anderson, 2003), all other factors being equal. Today more than ever it is possible to reuse existing software libraries that cover the basic functionality of a GIS application, and in many cases source code is available so that these libraries can be modified or extended by system administrators in-house. Examples of these libraries are Geotools (www.geotools.org) and Terralib (Camara, 2000), which are distributed under the Lesser General Public License (LGPL), meaning they are valid both for open source developments and also for commercial purposes.

In the following sections we describe in greater detail the proposed package-based architecture, followed by a pilot application implemented using Java WebStart (Schmidt, 2001) and integrating free software components as the package extensions to a central kernel.

ARCHITECTURE OVERVIEW

In this section we present a relatively simple solution that fulfils many, not all, corporate GIS needs. We modify the traditional concept of monolithic GIS application, and we redistribute it as smaller, interchangeable chunks to create a better fit to changing corporative environments, which among other things are connected via high-speed networks. The architecture proposed would not have been viable in most organizations a decade ago, as shown in previous studies (Anitto, 1994), due to the communication capabilities that are needed. The architecture has been defined to offer two key features: 1) inexpensive system maintenance and 2) scalability of the system in a natural, incremental manner, avoiding costly reengineering processes.

In order to achieve the desired flexibility, the system is divided into a kernel and a set of external packages. The kernel is considerably smaller, by a factor of 10, than the typical base GIS. Our packages (of the type used in the Perl, Python and Java programming communities) are larger functional aggregations than are GIS components such as MapObjects (Hartman, 1997). The kernel implements the main thread of the system, the basic user interface, and is in charge of assembling the GIS application at the client side. In the assembly process, the user selects and requests packages that are loaded and joined in order to build the final application. Each package implements a specific functionality and is connected to the kernel thanks to the adoption of well-defined interfaces (again, not necessarily OGC).

Software maintenance for a changing system is an important factor that we must take into consideration at the system design level. In this way and according to (Isnard, 2004), the administration of the maintenance in a specific environment simplifies this task. Here we assume that the system administrator handles the update and preparation of each package, its interfaces and necessary wrappers, so that end users do not need to program.

Corporate needs in GIS applications

After reviewing existing solutions for Corporate Geographic Information Systems (CGIS), we have pointed out some limitations overcome by the proposed architecture.

- Customization: Thanks to EUD principles, the user must be able to select and install only the specific packages needed to perform his/her particular work and not necessarily the full CGIS functionality.
- Extensibility: The system must be scalable according to evolving needs of the organization, allowing re-use of part or all of the existing system.
- Software distribution: Organizations operate in a decentralized fashion, and do business in many geographical regions, so they require distributed computer support. The software must be downloaded and installed across a network, to perform specific local installations.
- Maintenance: A centralized architecture helps in the administration of the system, because it allows updates to each independent piece (package) and interfaces with minimal cost.

General Structure

Traditionally GIS architecture follows a client-server schema (still rather monolithic, or bilithic), where the data processing takes place on the server side and the results are sent to clients for visualization. The question remains where to locate in a client-server schema, packages oriented to e-government or e-business with typical functionalities such as user validation and authentication. At the server side the system loses flexibility and introduces a performance bottleneck at execution time. On the client side the system overloads communication between server and clients due to redundant functionality sent to the multiple clients. Taking into consideration these aspects, the schema that has been adopted corresponds with a three-tier software architecture (see figure 3) by adding a middle new tier located between the client and server.

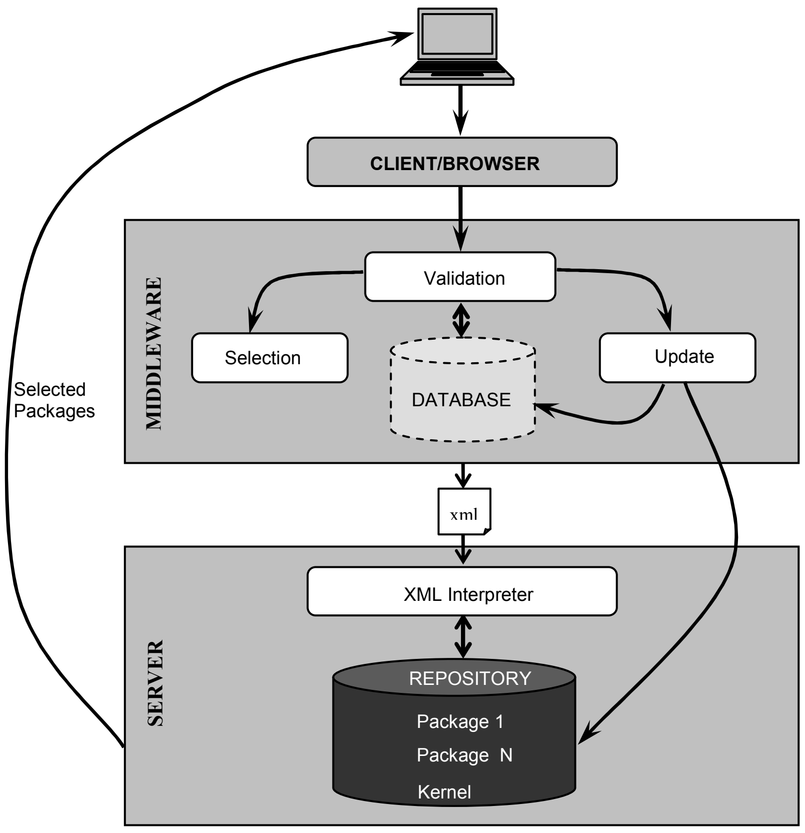


Figure 3: Architecture proposed for selecting and loading packages.

The package architecture exploits the main benefits of the web as a transfer protocol (ubiquity, portability, reliability and trust) for delivery of the different pieces that will conform each GIS application. The client layer is responsible for interaction with the user and for generation of queries sent to the middleware housing the user profile validation and authentication services. These requirements, in practical terms, are achieved with a web-based system architecture where the client is a standard web browser and the middleware is implemented as a web server, normally Intranet. The server layer, as software repository, contains the packages that will be sent to the client after processing the middle tier messages. The interchange of messages between the different layers is implemented with XML files (Aloisio, 1999).

In order to compose the final GIS application the main steps to be followed are:

1. Connection to the middleware layer through the client browser, the first time the user composes the GIS application and each time the GIS application configuration is updated.
2. Validation and authentication with user name and password. According to the user profile and package visibility privileges, the middleware shows a list of the possible packages (read here functionalities) that can be added to the GIS application kernal.
3. Selection of packages (functionalities) that the user wishes to install.

4. XML configuration. After receiving the result of the selection, the middleware dynamically creates a XML file that contains the user profile and package configuration information. This configuration file is sent to the server.
5. Server-side processing. The server processes the XML configuration file and sends the selected packages to the client. These packages are automatically launched to the client and the final GIS application is composed and initiated.

TESTING PROTOTYPE

In order to demonstrate the proposed architecture, we have implemented a testing prototype that follows the previous ideas. Although the architecture is not specific to any specific programming environment, in order to define a system as universal as possible, we have selected Java (Gosling, 1996). One of the more important features of the software products developed in Java is platform independence. Applying the concepts presented earlier to the public administration domain, the CGIS desired should be accessible to all the users without distinction to either hardware or operating system platform, again, to ease the load on system administrators who must support hundreds of users. As interoperability is a key factor for the proposed architecture this development language fits well with our needs. It is also no accident that many of the GIS-related software packages we have encountered are Java applications.

On the other hand software development should be based on well-known standard technologies, and this is where certainly adoption of OpenGIS/ISO interfaces helps considerably. Data formats should also be standardized so as to minimize reformatting needs between software packages, and also to facilitate the search and retrieval of data exploiting future Spatial Data Infrastructures (SDI). In this sense one of the packages currently being implemented for the testing prototype, accomplishes one of these functions with the inclusion in the CGIS prototype of a Web Map Service client (WMS OGC). The GIS application should be interoperable with other servers and facilitate data sharing among different governmental agencies. Fortunately many of the software packages encountered in the GIS free software community support a small set of well-known data formats.

According to the structure shown in figure 3, the final CGIS prototype is divided in three main tiers:

Client

The client tier is represented by a standard web-browser and allows the user to send different requests to the middleware through the network. This is a key element of the CGIS in order to define the architecture in an interoperable web-based context framework. Before accessing the functionalities offered by the CGIS, the user must be validated through the client interface, and so general security issues are included in order to support this authentication.

Middleware

The implementation of this intermediate tier has been carried out using Java Servlet technology. User access to the functionality provided by this tier is handled through the use of a standard web browser. The typical functions of this layer are the validation and authentication of the users as well as the selection of the packages to be installed at the client side. The users of the CGIS are registered in a PostgreSQL database located at the central server. In the authentication process this database is checked in order to identify the user. Depending on user and package privileges, the system displays to the client a list of available packages and their properties. In the case of users with administration privileges, the capability for uploading and/or updating packages to the system is also activated.

Besides name and privileges, each package carries some related metadata. These correspond to the author name, a brief description of the package functionalities, the date of creation, the version, size, etc. All these data provide information about the packages shown to the user at the selection process. In order to distribute the packages in an optimal way, they are compressed in jar files.

After the user has selected a set of packages to be installed, the middleware creates dynamically an XML configuration file, containing information about the packages to be sent to the user. The communication protocol selected for the testing prototype is JNLP.

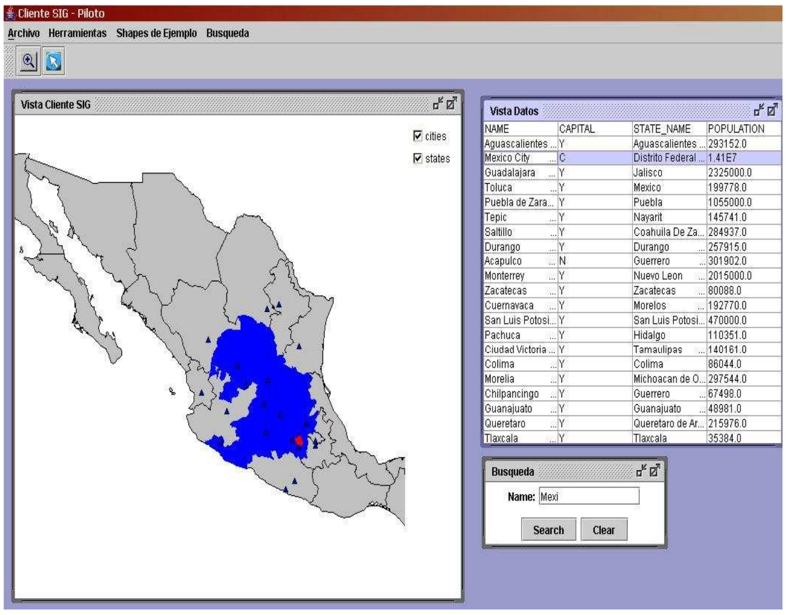
Server

The server receives and processes the XML configuration file sent by the middleware, and afterward sends the selected packages to the client. In order to perform these actions the JNLP communication protocol also has been used. This protocol channels Internet with the user system using low-level TCP/IP protocols, and sends the jar files across the web. The JNLP operations are performed by Java Web Start platform that must be installed (once) at the end user's client machine.

The JNLP protocol is implemented with the Java programming language platform and provides rather stable security mechanisms such as a signature associated with each package. It is the decision of the client to accept or not the download of a package depending of the validity of the signature. This feature is not important for restricted networks such as intranets, but can be crucial for the use of the CGIS in open network as it is the case of Internet.

GIS-Application

The GIS client application (client instantiation of the CGIS) is composed of several packages; the kernel and other additional packages that enrich the functionality of the basic GIS framework (a basic graphic user interface shown by the kernel). In principle, the kernel implements a basic graphic environment and is also in charge of the loading process of the external packages. It is divided in the following parts: main view, data view, menu bar and tool bar.



NAME	CAPITAL	STATE_NAME	POPULATION
Aguascalientes ...	Y	Aguascalientes ...	293152.0
Mexico City ...	C	Distrito Federal ...	1.41E7
Guadalajara ...	Y	Jalisco	2325000.0
Toluca ...	Y	Mexico	199778.0
Puebla de Zara...	Y	Puebla	1055000.0
Tepic ...	Y	Nayarit	145741.0
Saltillo ...	Y	Coahuila De Za...	284937.0
Durango ...	Y	Durango ...	257915.0
Acapulco ...	N	Guerrero ...	301902.0
Monterrey ...	Y	Nuevo Leon ...	2015000.0
Zacatecas ...	Y	Zacatecas ...	80088.0
Cuernavaca ...	Y	Morelos ...	192770.0
San Luis Potosi...	Y	San Luis Potosi...	470000.0
Pachuca ...	Y	Hidalgo	110351.0
Ciudad Victoria ...	Y	Tamaulipas ...	140161.0
Colima ...	Y	Colima	86044.0
Morelia ...	Y	Michoacan de O...	297544.0
Chilpancingo ...	Y	Guerrero ...	67498.0
Guanajuato ...	Y	Guanajuato ...	48981.0
Queretaro ...	Y	Queretaro de Ar...	215976.0
Tlaxcala ...	Y	Tlaxcala	35384.0

Name: Mexi

Figure 4: GIS-application composed for the analysis of vector data

Each of these parts is implemented independently as a class. These classes show a public interface that allows access to the objects or the implementation of internal operations. Following these interfaces, the packages are able to gain access to the kernel of the system. This connection is possible thanks to the XML connection files, which match a defined schema (DTD).

Figure 4 shows a selected configuration of the GIS-application. We observe that different packages of vector data visualization and basic functionality have been loaded. Specifically, we are visualizing a map of the Mexican states and its corresponding data table. The operation shows the search of a city, specifically Mexico D.F. The result is highlighted in the data table and in the main view.

CONCLUSIONS

In this paper, we examine some of the limitations of GIS applications used in corporative environments, which in many cases either too monolithic and general or too atomic and programming based. We concentrated on some crucial issues of integrating component-based software development and three-tier dissemination structures. These ideas are the result of previous investigation into the uses of GIS software in the public administration and the possibilities of open source software integration in this field. We have attempted to find the appropriate design for corporate environments in an effort to minimize maintenance costs, while at the same time promote the interoperability and flexibility of the resulting software framework.

We proposed an adaptive, package-based architecture that allows the optimum distribution and installation of the GIS application, package by package, where each user is able to customize his/her application interactively. The software architecture has been validated through the implementation of a testing prototype. This development has been a key aspect in order to improve and demonstrate the design features. The architecture is universal, not strictly related to any protocol, programming language or platform, though some characteristics of the selected implementation language, for instance being a multiplatform language, make the system even more universal. In particular, in the validation implementation Java WebStart was utilized with success.

Finally, the described architecture opens a framework for the development of GIS. With the inclusion of independent packages in a central server, different needs of the corporate environment can be supported. As future work we propose the implementation of new packages that extend the existing testing prototype. It is important to make an effort to incorporate open source libraries such as GeoTools 2 and Terralib that allow the creation of new packages with minimal effort; on the other hand the system can be extended in the middleware layer. We propose to add functionality for monitoring, and controlling downloads of the different users in order to produce useful statistics for administration purposes.

Acknowledgements

The work described here is sponsored in part by Spain Ministry of Science and Technology grant TIC-2003-09365-C02, and by the Generalitat Valenciana (CIT).

BIBLIOGRAPHY

Aloisio G., Millilo G., Williams R.D, 1999: An XML architecture for high performance web-based analysis of remote-sensing archives. Future Generation Computer Systems, vol. 16, pp. 91-100.

Anderson G., Moreno-Sanchez R., 2003: Building Web-Based Spatial Information Solutions around Open Specifications and Open Source Software. Transactions in GIS, Vol. 7, 447.

Anitto R. N., Patterson B. L., 1994: A new Paradigm for GIS data communications. URISA Journal, Milwaukee, pp. 64-67.

Brown A. W., 2000: Large-Scale, Component-Based Development. Prentice Hall PTR.

Camara G., et al.: Terralib: Technology in Support of GIS Innovation. II Workshop Brasileiro de Geoinformática, Geoinfo2000. Sao Paulo (2000) http://www.dpi.inpe.br/nsf-cnpq/terralib.pdf

Gosling J., McGilton H., 1996: White Paper: The Java Language Environment. Java Articles. http://java.sun.com

Hartman R., 1997. Focus on GIS component software. Onward Press, Santa Fe, New Mexico.

Isnard E., Perez E., and Galatescu A., 2004: MECASP – An Environment for Software Maintenance and Adaptation. ERCIM News, No.58, pp. 45-46.

McKinley P., Masoud S., Kasten E., Cheng B., 2004: Composing Adaptive Software. Computer IEEE Computer Society, vol 37 (7), pp. 56- 64.

Morch A., 1997: Three Levels of End-User Tailoring: Customization, Integration, and Extension. In: M. Kyng & L. Mathiassen (eds.): Computers and Design in Context. The MIT Press, Cambridge, MA pp. 51-76

Morch A., Stevens G., Won M., et al., 2004: Component-based technologies for end-user development. Communications of the ACM, Vol 47 (9), pp. 59-62.

Schmidt R.: Java Networking Launching Protocol & API Specification, 2001. Java Articles http://java.sun.com