WEB-BASED DATA MANAGEMENT

Matthew B. Jones

National Center for Ecological Analysis and Synthesis, 735 State St., Suite 300,
Santa Barbara, CA 93101

Abstract. Data management techniques for integrating World Wide Web (web) publishing with data storage systems are important in furthering innovative and powerful insights in ecology, mainly through improved data exchange and collaboration. This chapter discusses the benefits and disadvantages of using the web for data management applications, and describes four categories of technological solutions that can be employed to integrate database systems with web distribution.

INTRODUCTION

Data management practices at the National Center for Ecological Analysis and Synthesis (NCEAS; *http://www.nceas.ucsb.edu*) are used to promote our mission of "advancing the state of ecological knowledge." Many projects at NCEAS involve the acquisition, synthesis, and analysis of data from multiple, distributed data sources, as well as remote collaboration on projects before and after events are convened physically at NCEAS. Several other ecological groups have verified the need for remote access to large and long-term ecological data sets (Gross et al. 1995, Michener et al. 1997). To promote the widespread sharing of ecological data, and to improve remote collaboration activities among our research scientists, NCEAS has researched, developed and is implementing a variety of techniques for managing data using the World Wide Web.

Objectives of this chapter are to: 1) present an overview of the costs and benefits of developing and using web-based tools for data management; 2) examine several common technical approaches to integrating databases with web access; and, 3) conclude with some implementation guidelines that have proved useful in systems developed at NCEAS.

BENEFITS AND DISADVANTAGES

The relative merit of using the web for data management applications can be distilled to a tradeoff between the portability and accessibility of Internet-deployed applications versus the potential costs of developing and maintaining web services at a site (Table 1). Web-based applications generally work across all computing platforms in an organization, either through a common web browser client, or through Java® applets. Both of these approaches can be deployed locally at a site, or they can be made accessible over the Internet for the broader community to access. In addition, using the web as the client interface for data management tools allows developers greater flexibility in their choice of scalable database solutions deployed on the server because interface considerations are divorced from storage and application logic needs. This separation of interface from application logic and data storage also permits application developers to migrate to new and improved technologies on the server as they become available without any change in the client-side interface that the user experiences. Finally, using the web for data management affords the obvious advantages of easing the process of sharing and distributing data with collaborators, the general public, and archive centers.

Although there are clear benefits to using the web for data management applications, several potential disadvantages also exist. First, the web is a relatively new and immature technology, and so the development tools available for creating web applications are, compared to other development areas, feature-poor and difficult to use. For example, the types of Rapid Application Development (RAD) tools available to C/C++ programmers are just beginning to emerge for web

and Java® based development. Another facet of the technology's immaturity is reflected in the simplicity of the interfaces that one can build using HTML forms. Developers are limited to a small set of graphical widgets for use in presenting a user interface. Finally, like other complex technologies, web-based applications can require a high investment in software, as well as maintenance costs for the software and personnel for software administration.

Table 1. Costs and benefits of web-based data management.

Benefits	**Costs**
Cross-platform interfaces	Potentially high development and training time
Internet deployable	Potentially high software investment
Scalable database backend	Maintenance costs
Independent interface allows database migration	Interface simplicity / immaturity
Easier data sharing, interchange, and archive	

APPLICATIONS

The uses of web-based solutions in ecological data management are many and varied. One of the most obvious is to use the web as a mechanism for the distribution of existing data sets and their associated metadata. However, one can also use interactive web applications to create data entry forms for the collection of metadata and data, and to query and retrieve metadata. Data stored in a database can be converted to structured markup languages for interchange of data (e.g., XML) and for data presentation (e.g., HTML). Sophisticated query and visualization tools can be developed that give users a mechanism to remotely query data, find the subset that interests them, and then perform remote processing operations on those data. Remote processing functionality that is of interest to users includes quality control processing, data subsetting and aggregation, generation of descriptive and summary statistics, and generation of graphics for data visualization. Providing these simple analytic and visualization tools via a cross-platform, simple interface like the web empowers users to explore and use data that otherwise might be inaccessible.

WEB-DATABASE INTEGRATION TECHNIQUES

Overview

A wide variety of techniques exist for implementing the communication and data transfer mechanisms between web servers and data storage systems. The web is a client/server paradigm, so there is a tension between centralization of functionality at the server and distribution of functionality to clients. The most prevalent software solutions today do essentially all processing on the server side, and leave the clients with user interaction and display of information. However, recent advances in programming technologies (i.e., Java®) have blurred the roles of the client and server and promise to permit more advanced processing on client computers in a portable fashion. Although a number of platform or operating-system specific solutions exist, I have concentrated here on technologies that can be implemented cross-platform because the web was designed as a platform-neutral communication mechanism.

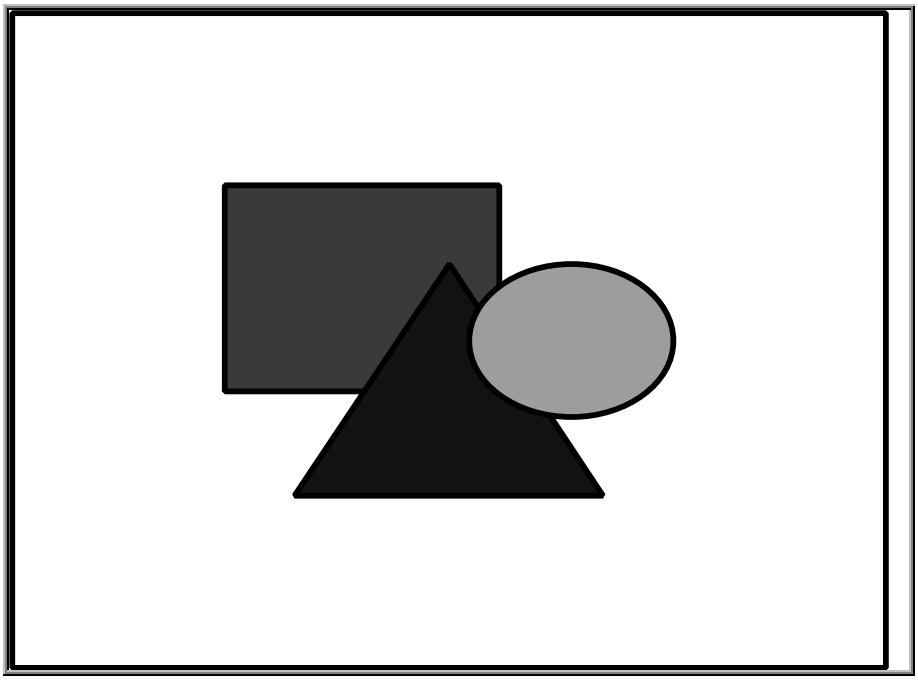
The techniques for web integration that are commonly employed can be broken into four general classes: 1) ASCII-oriented solutions; 2) Template parsing solutions; 3) Transaction monitor (middleware) solutions; and, 4) Java® applet solutions. The following sections will outline the basic features and benefits of each approach. Portability, scalability, ease of deployment, interface maturity and flexibility, cost, and client-processing capability are all considerations in evaluating the appropriateness of each technique for a particular application.

ASCII-oriented solutions

The widespread adoption of the ASCII standard as a universal character set has obvious advantages in terms of cross-platform portability and ease of deployment. The simplest case of web-based access to a data store, and one used as a foundation in many of the other techniques, is the delivery of static text documents from server to client over the Hypertext Transfer Protocol (HTTP). Web browsers (i.e., HTTP clients) are generally built to interpret and format the special type of ASCII documents known as HTML (Hypertext Markup Language) documents, but can in fact receive any type of data using this transport mechanism. To increase control over how data are delivered, many implementations add a processing script as an intermediary between data files and web server delivery of those files, making the process dynamic. The processing script can perform a number of tasks, including query processing, generation of formatting information like HTML code, data aggregation, etc. The script that executes often works by examining an ASCII text file that contains the data to be searched or processed. After determining which data are appropriate, the script formats the information and returns them to the web server. These processing scripts generally conform to the Common Gateway Interface (CGI), a standard that defines the mechanisms by which a web server can execute and communicate with processing scripts (Gundavaram 1996). CGI scripts can be written in most languages, including perl (common on UNIX), C™, C++™, Visual Basic™, and many others. CGI is simple to implement, inexpensive, and fairly easy to maintain, but generally does not scale well as the transaction load increases. In addition, the user interface elements available through the HTML "forms" specification are sometimes limiting, as is the lack of client side processing when CGI is used on the server.

Figure 1 illustrates the architecture of typical client/server transactions involved in delivering data via the web and CGI mechanisms. A web client requests, either via a URL or an HTML form, a set of data from the web server (solid arrow). The web server finds the requested file itself from the filesystem and returns it to the client (dashed arrow), or it executes a script, passing query information to the script via the CGI mechanism. The script executes and retrieves information from the filesystem according to the query parameters it received from CGI. When processing completes, the script sends the data (usually formatted in HTML) back to the webserver via CGI, and the webserver in turn sends the data back to the client that first made the request. This is a 2-tier client server solution, as the client and server generally reside on two different hosts.

Figure 1. Architecture of CGI text processing solutions.

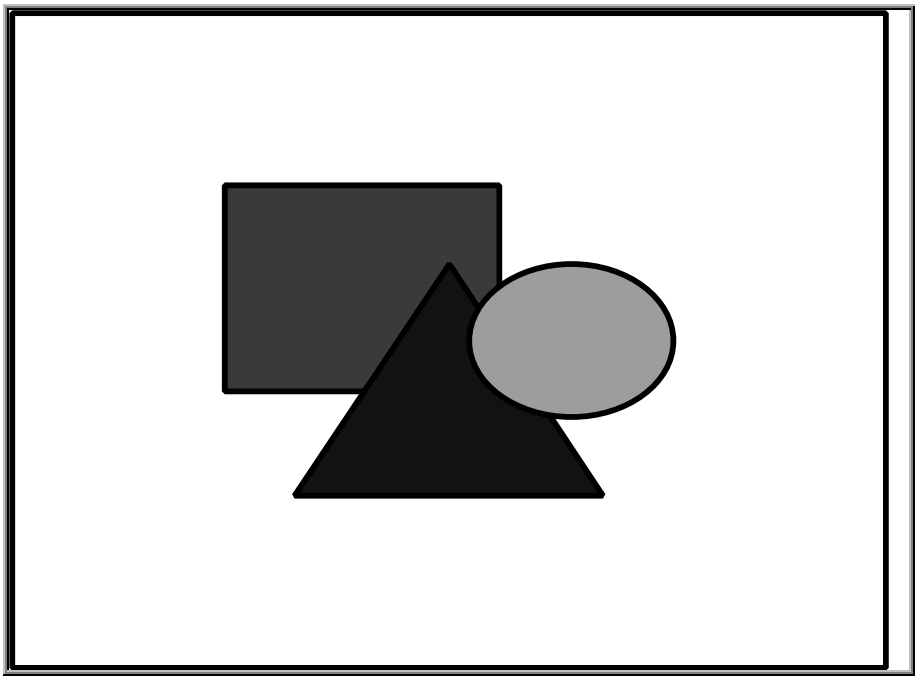


Template parsing systems

To improve access to database systems, several vendors have created systems for directly embedding database specific commands into HTML and other text files. A vendor-supplied program parses the HTML file and extracts the embedded commands, sending a database query to a database management system. The results that are returned from the query are interspersed in the HTML file according to formatting instructions, and the dynamically generated results are returned to the web server, which sends them to the client. This mechanism is similar to the ASCII database solutions described above, except that a proprietary language is used to embed commands in the HTML file that drives the query and formatting processor. Also, template parsing systems generally connect to relational database systems, and therefore they have the advantage of simplifying database integration. They are easy and powerful mechanisms for accessing a database, but generally lack scalability, don't contain the procedural functionality of more generic programming languages, and still are limited by HTML form interface elements and a lack of client-side processing. Examples of these systems include Allaire's Cold Fusion® and Microsoft's Internet Information Server® / SQL Server® combination; the Allaire product has the advantage of working with any Open Database Connectivity (ODBC®) compliant database -- a database interoperability standard -- and any CGI compliant web server, rather than being limited to specific products.

The architecture of template parsing systems (Figure 2) is similar to CGI / ASCII database systems. Again, a web client requests information and the web server passes the information via CGI to the template parsing program. The template parser retrieves the HTML template with embedded database commands, parses out the commands, and then makes a database connection (often ODBC) in order to execute those commands. The query results returned from the database are formatted by the parsing program and returned to the web server, which returns the dynamically generated document to the client.

Figure 2. Architecture of template parsing systems.

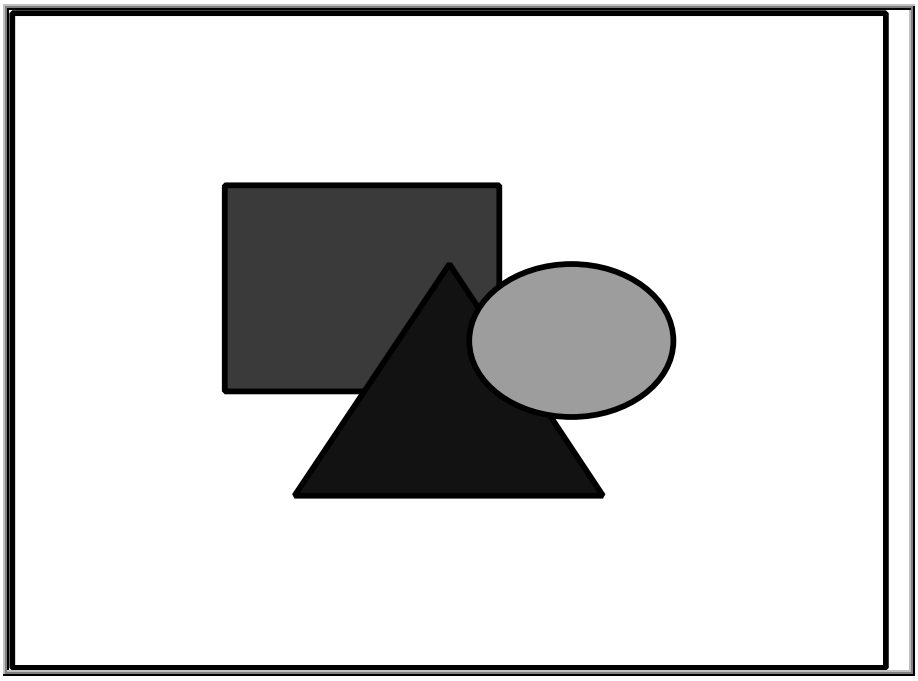


Transaction monitor systems

A further extension of these concepts arises in the class of solutions called Transaction Monitor (TM) Systems (sometimes called “middleware”). Transaction Monitor software usually implements a 3-tier architecture where the client and database each reside on different systems than the transaction monitor, and the transaction monitor plays the role of mediating transactions between the requesting client and one or more data providers that can be distributed across multiple other hosts. This architecture is extremely flexible and scalable because it allows many backend database systems, each potentially running different database software, to participate in a transaction over the web. In addition, the transaction monitor can actively poll the available server systems and determine which has the most available processing resources, thereby increasing performance and distributing computational load across the server database systems. Examples of systems that can implement a transaction monitor system include Oracle's Web System®, and Microsoft's Transaction Server®.

An example transaction monitoring architecture is illustrated in Figure 3. As usual, a web client makes a connection using HTTP to a server, which then launches the transaction monitoring (TM) software. The gateway between these systems can be CGI, but more often it is a proprietary interface that maximizes performance. The TM system then distributes query requests to one or more relational database systems on the same or different hosts (n-tier). Again, the gateway between the TM system and database systems are generally high-performance, proprietary drivers provided by the TM system. In addition, the database systems themselves often store the application logic and formatting instructions in stored procedures, rather than having to parse text transmitted via the web server gateway (e.g., CGI).

Figure 3. Architecture of transaction monitor systems.



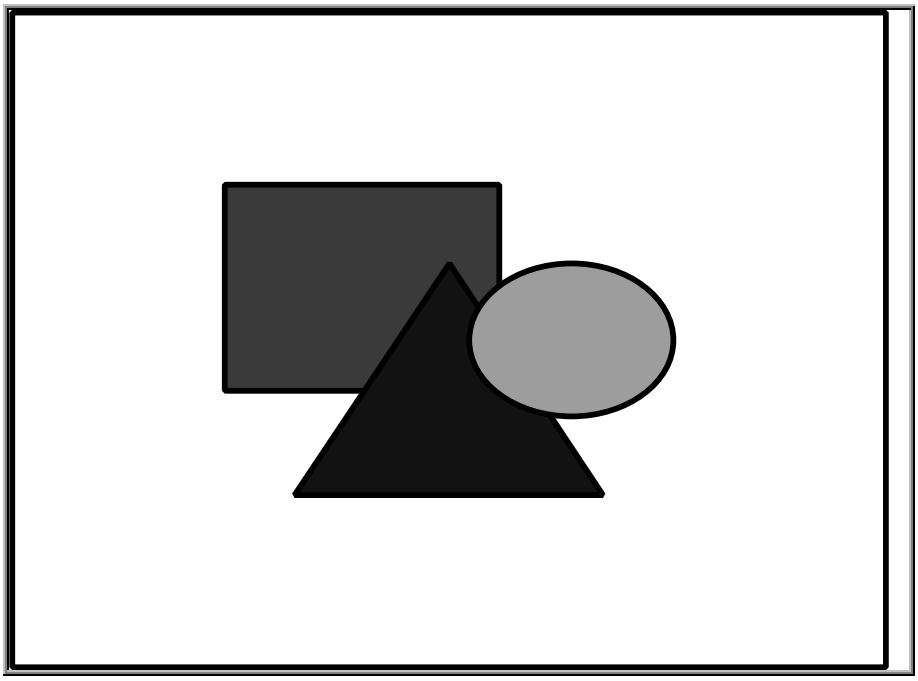
Integration using Java® and JDBC®

The Java® programming language has a library called "Java Database Connectivity®" (JDBC), which provides a platform and database independent programming interface to access multiple distributed databases of varying types. In using this system for integrating databases with web sites, one develops Java® applets that are delivered over a web connection (HTTP), and then the applets execute on the client machines. This mechanism is by far the most flexible because it allows the programmer to design an n-tier database system with connections to many database systems, all without specialized, expensive middleware software. Because the applet runs on the client machine, it allows full freedom in client side processing for field validation and interface fine-tuning. There are two principal disadvantages: Java® is a lower level language than others described here and therefore is substantially more complicated to use for interface development; and Java's performance is still much lower than many natively compiled interface-building systems. However, for most interface activities, performance is not particularly demanding and Java® will usually allow more responsive interfaces than HTML does. Other systems like Microsoft's ActiveX can be used to implement similar systems, but they lack the basic advantage of all of the systems described here: interoperability across virtually any operating system. Java® applets are employed in this way in several commercial systems, including SAS' Intr*Net® product and many middleware systems such as Symantec's dbAnywhere®. JDBC® drivers exist for most major database systems, including Oracle®, Sybase®, SQL Server®, and others.

A typical architecture for using Java® and JDBC starts with a web client requesting an HTML page that has an embedded Java® applet (Figure 4). The web server delivers the applet to the requesting client (a potentially time-consuming process), and then the client executes the Java® applet, ending the interaction with the web server. The Java® applet uses JDBC calls to open up separate TCP connections to one or more relational database systems, independent of the web server. It then communicates with these database systems using JDBC calls to query and update data, while displaying the results in a custom developed user interface. This type of mechanism allows substantially more flexibility in implementation than any of the other systems,

at the cost of development time. The complexity of designing application logic for a Java® applet to manage one or more database connections and an easy-to-use interface should not be underestimated, but neither should its potential power.

Figure 4. Schematic of a Java®/JDBC architecture.



CONCLUSIONS

The variety of mechanisms described here allow everything from simple, easily implemented web-database communication to high end, scalable solutions for critical applications. The categorization that I developed was a means of simplifying a continuum of overlapping, non-exclusive technological solutions, and should be interpreted as such. For example, many transaction monitor systems may use CGI gateways, and Java® solutions may make more HTTP connections than indicated. Nevertheless, the basic features of those systems are used as indicated.

When designing a mechanism for web-database integration, one must weigh the relative strengths and weaknesses of the different approaches outlined above for a particular application. If the application is relatively local in scope and small in scale, it will probably be simplest to use the CGI-based ASCII approaches. For more complex applications, and for applications where scale and performance are critical, some of the more complex approaches outlined here, such as transaction monitor systems, may be appropriate. Finally, where substantial control of processing on the client computer is needed, and where portability across computing platforms is important, custom-designed Java® applications and applets become beneficial choices.

In implementing and researching these solutions, I have found a number of general guidelines useful to keep in mind across all of the systems. First, as soon as one attaches a computer to a network, and especially when one provides access to data over the Internet, security concerns arise. Writing both CGI scripts (in any language) and Java® programs has inherent risks; one must carefully examine the mechanisms by which user input is validated and checked before it is used to execute programs on the server system, or you may inadvertently grant full access to a database or operating system (see Garfinkel 1997). Second, although some aspects of web-

database integration seem simple, full scale integration is much more difficult to design and implement; conservatism in estimates of development time help to make projects successful. Designing a modular system in which each module has utility before the entire system is completed can help in this regard, as well as making it easier to upgrade modules as new technologies arise. Third, all of these mechanisms for integration allow a clean separation of user-interface from data storage; by designing your applications this way you can upgrade backend storage systems when the need arises without impacting the user's method of interacting with data.

In the end, these technologies are only useful to ecological data managers when they improve the quality of science in the discipline or open up new areas for research. At NCEAS we hope that the integration and synthesis of data will allow new insights into the structure and function of ecological and evolutionary systems. Our development of data management technologies is guided by our need to synthesize data from multiple sites, or data arriving in many formats, as well as a desire to exchange data with colleagues. This paper represents a synthesis of technological solutions that ecological data managers may find useful in their own efforts.

ACKNOWLEDGMENTS

The ideas in this paper were substantially improved through discussions with Mark Schildhauer. This work was funded by the National Center for Ecological Analysis and Synthesis, a Center funded by the National Science Foundation (Grant #DEB-94-21535), the University of California - Santa Barbara, and the State of California.

LITERATURE CITED

Garfinkel, S. and G. Spafford. 1997. Web Security & Commerce. 1st Edition. O'Reilly and Associates, Cambridge, MA.

Gross, K.L., C.E. Pake, E. Allen, C. Bledsoe, R. Colwell, P. Dayton, M. Dethier, J. Helly, R. Holt, N. Morin, W. Michener, S.T.A. Pickett, and S. Stafford. 1995. Final report of the Ecological Society of America Committee on the Future of Long-term Ecological Data (FLED). Volume I: Text of the report.

Gundavaram, S. 1996. CGI Programming on the World Wide Web. 1st Edition. O'Reilly and Associates. Michener, W.K., J.W. Brunt, J. Helly, T.B. Kirchner, and S.G. Stafford. Non-geospatial metadata for the ecological sciences. Ecological Applications 7:330-342.