IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4725

Complex Decision Support in Web Time

Daniel Brandon, Ph.D., PMP
Christian Brothers University
Information Technology Management Department
650 East Parkway South
Memphis, TN 38104
email: dbrandon@cbu.edu

OVERVIEW

Over the last several years there has been a move towards migrating and/or re-engineering classical decision support systems from traditional mainframe or client/server systems to web based systems. This move, or better called a "push", is driven by the need to quickly reach larger audiences and for a lower "total cost of ownership" [McGee, 2002]. The need to reach larger audiences is driven by a need for both larger intra-company users and lately a need to reach users outside of the company who may be paying a fee for the decision support information.

An environment, where the decision support users have a need for only internet browser software on their computer, forces a total redesign and new implementation of computer decision support systems in able to meet the response time and possible bandwidth constraints in this new environment.

This article discusses the issues involved in the re-deployment of decision support applications to the web environment and presents alternative approaches. We also present a "framework" for web based decision support applications based on our research and development. This framework, while not novel in its components, may be somewhat unique in the manner the component technologies are integrated. We also describe the results achieved for one common and classical type of decision support.

ISSUES

Consider the information flow depicted in Figure 1, which is typical of many decision support type of applications. Information is taken from possibly a number of "data stores", filtered and sorted, and merged. In textbook terms, information is data endowed with relevance and purpose typically by organization and appropriate processing. Further "business rules" may be applied, then the information goes thru some type of decision analysis for the conversion from "information" to explicit "knowledge". Next further business rules may be applied, and then the knowledge is presented to the user for study and possibly interaction.

Today's typical data stores are relational databases, data warehouses, sets of "standard" files, or possibly information in XML format. National, state, and local data is typically available in standard files such as GIS (Geographic Information Systems) type data or government financial type data. The extraction process may be expressed in SQL (Structured Query Language) or via logic embedded in specific extraction programs.

The analysis step might be a statistical process, an expert system, a neural network, a genetic algorithm, or some other type of decision application previously written to run in a mainframe or client/server environment. In classical decision support software systems, the entire hardware/software process was under the control of the managing/owning organization and typically took place within one network/vendor domain. The controlling organization could install whatever software pieces that were necessary on either the mainframe/servers or on the clients (PC/terminal); and that software might include business rules, user interface software, or other "middleware".

Common examples of the classical approach might be such processes as extracting data form a relational database and loading a predefined spreadsheet (such as Excel) with updated information to run a new analysis (using existing formulas and formatting). Another example would be extracting and merging data from census and other demographic or financial files into a statistical package (such as SPSS) to utilize one of the decision support algorithms therein. Sill another example would be extracting data from a data warehouse and updating an EIS (Executive Information System) system for a current analysis.

In the web environment, the organization has no control over the user's hardware or software, and no control over the network. The only requirement is that the user have an internet browser. The user's PC may be Microsoft Windows's based, Apple, Linux, or other client operating system. The browser software might be Microsoft Internet Explorer, Netscape, or other. The user may have a new fast PC, or an old slow one. His connection to the web might be over POTS (plain old telephone) or high speed cable or DSL.

Relating these issues back to Figure 1, we see that an organization only controls one end of the flow. Also the division of work in terms of which tasks will be done on the organization's servers and which will be done on the client's PC needs to be investigated perhaps both in general and for each type of decision support application. The ideal situation would be to do as much work on the client's PC as possible without compromising the quality of the services (response time and accuracy of results) or the security of the application (or other organization assets).

Our early results at attempts to build responsive web based decision applications were not successful. We tried to use essentially the same process and software applications for the web that we had used in the client-sever environment (namely extracting data and loading it into our application of choice), and simply used the client to display results via simple HTML. For example, a browser would request some type of analysis of data. The data would be extracted from a relational database and feed into a standard analysis program (such as Excel or SPSS) via ODBC, OLE/OB, or some other linking middleware. When the program finished processing, the results were relayed back (again via some linking middleware) then formatted in plain HTML and sent to the client. The time involved with the initiation of separate server processes and the complexity (and susceptibility to errors) of the linking middleware causes major delays and often "lock-ups".

Figure 1

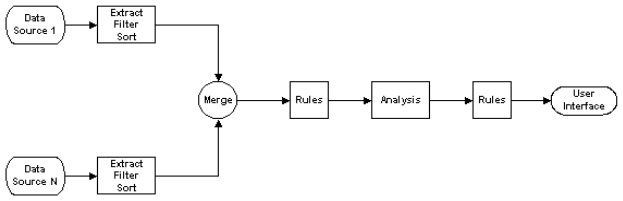
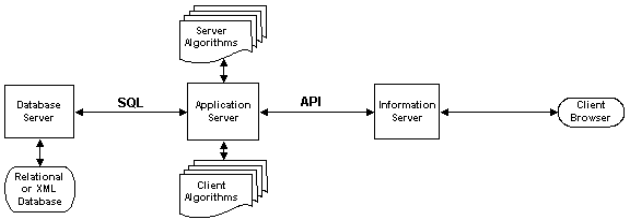


Figure 2



WEB DECISION SUPPORT APPLICATION FRAMEWORK

It was decided that a rigorous "decision support application framework" was needed which would provide guidelines on the design and implementation of web based decision support applications. The goals were to remove the delays and instability in trying to link programs that were never originally designed to be connected on-line for the internet. Another goal was to move processing (where possible) to the client's PC to minimize the load on the server thus minimizing the response time for all users.

Figure 2 shows the design of the general "Web DS Framework". It involves a database server to interact with a relational database, XML dataset, or data warehouse. Typical products used here would be typically Oracle, MySQL, DB2, SQLServer, or even Access (for systems with few database writes). An application server would communicate with the database server via SQL (and/or stored procedure calls). Typical products here would be Cold Fusion, Microsoft Active Server Pages, Java Server Pages, or Java Servlets. These products communicate via the Information server via a defined API, so there is a very intimate connection without "middleware", and without additionally created processes for each client request (as is true with CGI processing), and without per client connections to the database ("connection pooling"). The application server and the information server (typically Microsoft IIS or Apache) are usually on the same physical server or in a "cluster arrangement". The database server is typically on a separate physical server, but may be on the same physical server as the other two servers.

Instead of relying on separate classical applications for the analysis, each analysis algorithm would be written as a generic module that would be run on the application server or downloaded to be run on the client PC. The choice depends upon the nature of the algorithm and/or the size of the data set involved. With the power of modern PC's, it is believed that most decision support problems could be downloaded to run on the client PC. Since the choice of database products is varied, and the choice of application server products is varied, the server and client algorithms need to be implemented in a language that is not proprietary and one that is suited for the web. For the server algorithms, it is suggested that the Java language be used. All the application servers support it well except possibly Active Server Pages. For the client algorithms, it is suggested that JavaScript be used. Java could also be used and work with all clients, but on the client side one must be very careful on how Java is coded (and off course, we still have to deal with the political problems between Microsoft and Sun).

In terms of technical issues, the main difficulty is finding or making the necessary algorithms in Java or JavaScript. In the area of decision support, particularly with statistical algorithms, most are currently only freely available in FORTRAN and some in C. [Burden, 1997; Demmel, 1997; Press, 1992; StatLib, 2002; Watkins, 1991] There are a few companies that market Java source code for many such algorithms, but the cost is many thousands of dollars for each algorithm.

As concerns JavaScript for the algorithms that are going to download to the client with the HTML page, very few such algorithms have been developed. One of our concerns also was the adequacy of execution speed of JavaScript on the client and possibility of numerical stability (round-off error) problems. In addition the JavaScript language does not have direct support for multi-dimensional arrays and some math functions, so additional coding effort is required. For example to set up a two dimensional array, an array of arrays must be formulated with constructor functions as:

```
function Array2D (x,y){
      this.length = x;
      for (var ii = 0; ii < x; ii++)
                  this[ii] = new Array1D(y);
}

function Array1D (z) {
      this.length = z;
      for (var jj = 0; jj < z; jj++)
                  this[jj] = 0.0;
}
```

Figure 3 shows the detail production of the HTML page going to the client's browser by the "script" for the application server. Display standards are combined with application specific database information to produce an HTML display template, and business rules are combined with application and problem specific database information to produce the "arguments" for the JavaScript functions in the client JavaScript library. For security reasons, the JavaScript client library functions are encrypted. Also for security reasons the resulting HTML pages to the client browser have the necessary HTML headers (and JavaScript functions) to prevent local client caching (or framing) of the web pages and viewing or printing of any resulting HTML code.

Note that there are two portions to the JavaScript algorithms. The first part is the static encrypted JavaScript of the algorithms themselves that is stored in the "algorithm library"; these algorithms take scalar, vector, or array arguments. The second part is the dynamically generated JavaScript that builds the arguments for the algorithms; the dynamic generation is a function of the business rules and information in the supporting database.

EXAMPLE DECISION SUPPORT WEB APPLICATION

For a comprehensive test of the proposed Web DS Framework, we decided to try to implement a system that did predictions based upon a large database of past data using the common technique of multiple linear regression. We know this is a computationally difficult problem, since the solution complexity and time increases exponentially with the number of independent variables. We had been unsuccessful in implementing this on the web using a methodology of linking statistical packages to information servers (results were accurate, but delays and lockups were unacceptable).

The general multiple linear regression problem using matrix (linear) algebra involves the solution of the least squares minimization [Allen, 1997; Draper 1981; Fox, 1997; Seber, 1997; Weisberg, 1985]:

$$\mathbf{b} = (\mathbf{X'X})^{-1}\mathbf{X'y}$$

where **X** is a matrix of the data values for the independent variables, **y** is a vector of the

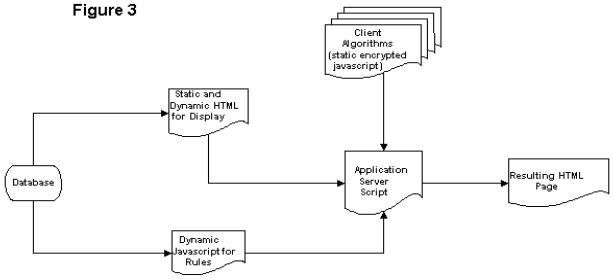
values of the dependent variable, **b** is the solution vector (weighting parameters),

-1 denotes the matrix inverse, and **'** denotes the transpose of the matrix.

Our test environment consisted of a single server running under Windows 2000 (NT) with Microsoft IIS as the Information Server. The database product was Microsoft SQLServer, and the application server product was Macromedia's Cold Fusion. The client PC's (which were contemporary and under 1 GHz) ran both Microsoft Internet Explorer and Netscape Communicator.

Business regression problems typically have between 2 and 7 independent variables, and the matrix solution time can rise exponentially with the number of independent variables and the choice of algorithm.

Figure 3



We first used an available JavaScript algorithm [Waner, 1998] that used a direct inverse/determinate algorithm. Server processing times, download times, and client processing times were all acceptable (under 1 second) until the number of independent variables went above 3 or 4. For 6 independent variables, the client processing time was measured in minutes. Next the algorithm was rewritten in JavaScript using a Gaussian elimination type method [Press, 1992; Spath, 1987; Watkins, 1991] and client processing time dropped down to under a second even for 6 or 7 independent variables. The accuracy of the solutions was also acceptable, with the 10 digit floating point precision of Javascript. The client regression portion is exemplified in a test program at:

http://www.cbu.edu/~dbrandon/regression/multiple_linear_regression.html.

CONCLUSIONS

The Web DS Framework has successful in building web based decision support applications that are considerably more reliable and responsive than traditionally used techniques. The stakeholders (developers and customers) been very pleased with the results of using the Web DS Framework on the numerically difficult test case described above. The framework has also been successful on several less numerically demanding decision support applications. We will be using this framework on other decision support applications in the future and expect continued success, although we anticipate some new problems may suggested modifications or extensions to our framework.

REFERENCES

Allen, M. (1997). **Understanding Regression Analysis,** Plenum Press.

Burden, R. and Faires, D. (1997). **Numerical Analysis**, Brooks/Cole Publishing.

Demmel, J. (1997). **Applied Numerical Linear Algebra,** SIAM.

Draper, N. and Smith H. (1981). Applied Regression Analysis, John Wiley & Sons.

Fox, J. (1997). **Applied Regression Analysis,** Sage Publications.

Mallach, E. (2000). D**ecision Support and Data Warehouse Systems,** Irwin/McGraw-Hill.

Marakas, G. (2003). **Decision Support Systems in the 21st Century,** Prentice Hall.

McGee, M. (September 16, 2002). Faster Data, Better Decisions, **Information Week.**

Press, W. et. al., (1992). **Numerical Recipes, The Art of Scientific Computing** (FORTRAN Version, Cambridge University Press,

Sauter, V. (1997). **Decision Support Systems,** Wiley.

Seber, G. **(**1977). **Linear Regression Analysis,** John Wiley & Sons.

Spath, H. **(**1987). **Mathematical Algorithms for Linear Regression,** Academic Press.

StatLib, Carnegie Mellon University, http://lib.stat.cmu.edu. 2003

Waner, S. and Costenoble, S. (1998). JavaScript Algorithm: multlinreg.html, [http://www.shodor.org/chemviz/tools/multireg/].

Thomsen, E. **(**1997). **OLAP Solutions,** Wiley.

Watkins, D. **(**1991) **Fundamentals of Matrix Computations,** John Wiley & Sons.

Weisberg, S. **(**1985). **Applied Linear Regression,** John Wiley & Sons.