IMPROVING AND UTILIZING WEB AND DATABASE INTERACTION

Christos Bouras, Vaggelis Kapoulas, Agisilaos Konidaris and Afroditi Sevasti
Computer Technology Institute-CTI, Kolokotroni 3, 26221 Patras, Greece
Department of Computer Engineering and Informatics, University of Patras, Greece
E-mails: {bouras, kapoulas, konidari, sevastia}@cti.gr

ABSTRACT

Web and Database Interaction is one of the growing trends in Web-site creation. This interaction can facilitate the needs of the online community in various ways. Until now databases were only used to hold large amounts of information needed for certain applications. Many of these applications are nowadays Web-based. The constant interaction that is needed by them is very demanding in terms of computer resources. In this paper we propose a technique that can reduce the amount of resources needed to run these demanding applications. We also introduce an alternative model according to which databases can be used for many other purposes in the process of Web and Database interaction.

KEYWORDS

Web-Database Interaction**,** Caching, Active Server Pages, Modularization.

INTRODUCTION

The issue of Web and Database Interaction (WDI) plays a significant role in the mainstream of corporate software development and applications for the Internet. As the demand for database services rises, high-performance and sophisticated WDI becomes a crucial issue. The most popular technological approach to WDI is through scripting languages, with the existent technologies differentiating in the level of Web server integration, the level of dynamic database integration, the extent to which code is generated automatically and the code structure. This paper makes a short reference to existent interaction approaches and proposes two major application enhancements for improving quantitative and qualitative features of contemporary WDI systems.

CURRENT TRENDS IN WEB AND DATABASE INTERACTION

A Web page, which merely provides information to users through access to files stored on a server, is known as a *static* Web page. Although very popular, static Web pages had a serious limitation: communication with the people browsing a site was one way only. Web pages of the next generation, where the reader of the page can interact dynamically with the server, are interactive. Presently, databases are being increasingly connected to Web pages. The technological approaches to the issue of WDI result into two different categorizations, the first being based upon the nature of the technology used and the second on the system architecture.

According to the first categorization (Kim, B.I., 1999), there are four general-purpose technologies for WDI: Common Gateway Interface (CGI), JAVA and JDBC, Database vendors' solutions and Third party vendors solutions. According to the second categorisation, the existent WDI system architectures can be divided into client-sided and server-sided.

The server-sided approach is older and more widespread. It allows the clients to start a CGI or NSAPI or ISAPI procedure on the Web server, connecting the clients to the database. The database responds, supplying the Web browser with the use of Dynamic HTML. This architecture is independent of the client's browser, therefore it is ideal for use over the Internet. The main drawback of the server-sided approach is that the client addresses the server, each time additional information is needed.
The client-sided approach is more recent and is based on the installation of a control or script on the client, which connects the client with the remote database. This way, the user does not have to constantly address the Web server in order to obtain new HTML code. JavaScript, VBScript and Dynamic HTML are the technologies most commonly used by this architecture. However, Java and Active X technologies, based on the DORMAT (download-once/ run-many-times) architecture, appear to prevail lately in the area of the client-sided architectures, as they offer more sophisticated mechanisms for WDI compared to the scripting technologies. All the above technologies restrict the amount of possible users to those whose browsers support them. Therefore, many Internet users are automatically excluded from client-sided technologies, whilst mainly Intranet users seem to benefit from them.
Independently from the categorization used, all the above technologies consist means for WDI, with each one of them applying in different cases, with different performance and functionality demands. Our approach does not interfere with the technologies' concepts, however suggests enhancements which in certain cases, under certain conditions appear to improve performance and quality of the services provided to the end user.

CACHING OF PAGES IN SERVER-SIDED WEB AND DATABASE INTERACTION

Our experience stems from a WDI system that was developed for the needs of the ELECTRA (European Electronic Information Centre for Adult Education) ISPO project (ELECTRA ISPO project, 1998). The system was developed using the server-sided architecture, the SQL Database Server (Hough, K., 1998) and the Internet Information Server (IIS) in combination with Active Server Pages technology for the WDI. The system is intended to Internet users, therefore the server-sided architecture was preferred (Fig. 1.). Active Server Pages are simple Web pages combining code in script language together with HTML code (Fedorov, A. et al., 1998). After the script ASP code is executed in the server, the browser-presentable part of the page is returned to the client.

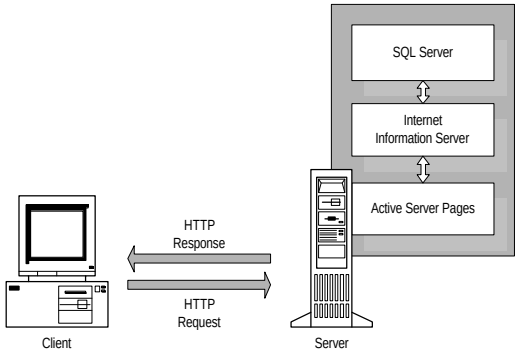

Fig. 1. System Architecture

The database implemented for ELECTRA is relatively extended and complicated. Our ASP pages are fully dynamic, meaning that they are created during the execution of the script parts of the pages. Due to the nature of the data stored in the database, the majority of the pages' elements require many SQL queries and HTML code to present the queries' results. During

the procedure of evaluating and improving the system, several interesting and challenging observations were made.

In the most common cases, where an ASP page contained one or two connections to the SQL server, performance was seriously degrading when the number of simultaneous users was increasing. More specifically, when ten users were trying to access the IIS simultaneously, the corresponding amount of connections resulted in hundreds of transactions with the SQL server (inserts, selects etc.) (Vaughn, W.R., 1998). Consequently, the database response time was increasing and the requested by the HTML part of the ASP pages' data was retrieved with delay.

The result in the client's browser was an annoying retardation in receiving and displaying the HTML pages. Of course, larger delays were observed in pages where almost all fields and elements are constructed using data retrieved from the database. In pages where the elements are static, corresponding to specific HTML code inside the ASP page, performance was not affected by the increase in the amount of users. In the worst cases, where the SQL server utilisation reached 99%, the impression in the clients' browsers was that the IIS was not responding anymore. It is important to notice that our performance evaluation was carried out over IP within the limits of an Intranet. It is obvious that the performance would be much worse if server delay was escorted by transmission delays of the Internet.

At this point, a brief evaluation of the structure concluding to the results above will be made, before proceeding to the improvement modifications proposed. In the case of server-sided WDI architectures, dynamic construction of the pages presented to the end user ensures that the data presented is accurate and up-to-date, since it is directly loaded from the database. At the same time, modifications in the database structure or contents do not require major reconstruction of the Web pages. Finally, pages are fully dynamic in time and space, allowing passing of variables and dynamic construction of succeeding pages, according to the users' interaction and the database contents. On the other hand, our experimental results demonstrated the performance limitations imposed by the dynamic ASP pages technique.

Based on the above analysis, we propose an enhancement to the ASP pages technology, which aims at preserving the advantages of the use of dynamic ASP pages while improving performance. The main idea is to develop a caching mechanism for storing locally (at the client) the pages sent from the server after the processing of the ASP pages, in other words the HTML part of the ASP pages. According to this convention, each ASP page will be processed by the server and sent to the client only the first time it is being referenced. All the sequential references to the same page from the client will be answered by references to the client's cache. The ASP page will be re-processed and re-posted by the server to the client only in cases of client cache miss. It is obvious that this technique will remove a substantial amount of workload from the IIS and SQL server, solving both performance problems of server overload and network traffic.

An issue that must be further discussed and investigated is the size of the client's cache. This is mainly a matter of how many pages the cache should be allowed to keep and not a matter of the exact storing space. Parameters which should be estimated here are the average of ASP pages referenced by the user in each session and the involvement of dynamic elements in the pages' construction. The size decided should ensure that pages are refreshed due to misses in medium time intervals, so that any changes in the database are reflected to the client. At the same time, the number of misses must be restricted so that the performance advantage of not addressing the server each time a page is requested can be exploited. A system with high cache miss rate will approximate a system with no such a caching mechanism.

Of course, what we propose here can be viewed as a hybrid solution to the problems described previously. Such an implementation ensures WDI, as it exploits the ASP technology for

constructing, from database contents, the pages reaching the client. However, some of the advantages of the ASP technology, are partially overlooked for performance reasons. The approach can prove to be really prosperous in cases where the database contents are not constantly changing. In that case, caching of pages works exactly as the fully operational ASP schema and there are only positive effects. In the rarer cases, where database contents are constantly changing, caching may result in loosening the WDI and presenting the user with obsolete information. In this case, the page reaching the client will be valid only the first time it is referenced. All consequent references will obtain the page from the client's cache, overlooking the fact that some parts of the page (those which are dynamically constructed from database queries) might contain outdated data.
The drawbacks of the proposed approach should be carefully investigated and dealt with. A more specific and firm, yet unexplored, solution is described in the 'Future Work' paragraph.

DYNAMIC WEB-SITE CREATION USING A DATABASE SCHEMA

In order for our proposal concerning dynamic Web-site creation using a database schema to be introduced, the term Dynamic Web pages must be clarified. A Dynamic Web page is a page that uses HTML only to make clear what the element that will appear on a page is. To make this clear an example will be given (Lazar, Z.P. and Holfelder, P., 1997). It is well known that the HTML tags <H1> and </H1> inform the Web browser that anything found between them must be of Header One format. A dynamic Web page in this case is a page that contains these tags but does not include static information for what should be between the two tags. Instead, a variable is found between them. In this example HTML is used to inform the browser of the type of information that will be shown but it does not convey any message about the information itself.
The main point that we want to make is that by using a database in the right way one can create fully dynamic, scalable, non-redundant and easy to edit Web pages. In order to achieve all of the above we propose the use of a certain database schema and certain Web programming tactics. The system includes two modules as shown in Fig. 2.

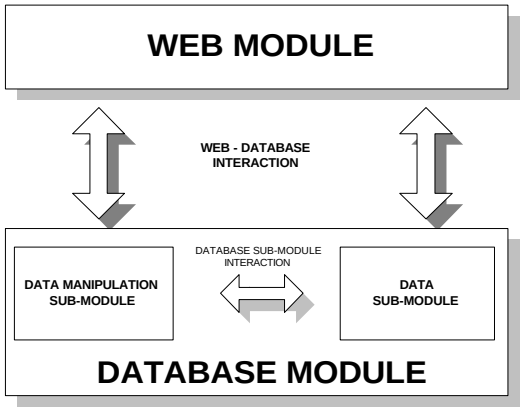

Fig. 2. Module interaction schema

The database module holds all the information that will appear on the Web site but also holds information about the information in terms of appearance. The Web module takes advantage of the database module and provides a sort of templated environment.
The database module of the system is responsible for containing the information that will ultimately appear in the client's browser. This information (images, text, applets etc) will be kept in the Data sub-module of the database module. The Data sub-module can be viewed as a "Raw" data storage module. This sub-module does not contain any information on how the

stored information will be used. The other part of the database module is the Data Manipulation sub-module. This sub-module contains the rules that designate the way that the stored data in the Data sub-module will be presented to the user and which part of the data must be presented. It also contains rules on how data is binded together. In the first case a presentation rule may be a rule that says "Any .GIF formatted picture that is contained in the GIFS table in the Data sub-module must be aligned left in the Web page". In order to apply these rules to any data stored in the Data sub-module we must introduce a database sub-module interaction process. This internal database sub-module interaction process can implemented by foreign keys that associate stored information in the Data Manipulation sub-module with information in the Data sub-module. In this way, any rule stored in the Data Manipulation sub-module "knows" to which data in the Data sub-module it applies. The Data Manipulation sub-module also binds data in the Data sub-module together. What we mean by binding data together is finding relations that must apply between certain data. To explain this here is an example of another rule: "A GIF formatted picture in the GIFS table must always be preceded by some explanatory text that will be found in the TEXT table". This rule binds together certain records in the GIF table with certain records in the TEXT table. To summarize the functionalities of the Data Manipulation sub-module we can say that it generally offers two services to the overall system. The first is that it "grants" topological characteristics to every record in the Data sub-module. Topological here refers to the where and how a certain record may be found in a Web page. The second Data Manipulation sub-module functionality is that it associates records in the Data sub-module.
The Web module has its own functionalities. The fundamental purpose of the Web module is to provide the conditions that will make certain rules in the Data Manipulation sub-module valid. These conditions may be user preferences, time dependent conditions and any other circumstance that a Web page designer may want to include in his pages. A Web module condition may be a conditional statement in a Web page that says :

Get User_IP
If User_IP starts with 150 then
Get rule 1 from data-manipulation submodule
Get rule 10 from data-manipulation submodule
Get rule 56 from data-manipulation submodule
End if

It is clear by looking at the example above that the Web module funcionalities are both the acquirement of user information or preferences and the decision based upon the information and the user preferences as to which rules are valid. The Web module handles the interaction process between the system and the user. It gathers information and preferences from the user and then requests certain rules from the data manipulation sub-module.
The system that we propose, as shown in Fig. 2., consists of two basic modules and two sub-modules. The main aim of the module separation is to distribute the systems logic into three discrete components. Generally the Web module consists of the client-system interaction logic and the decision-making logic. The Database Manipulation sub-module consists of the data manipulation logic and the Data sub-module has no logic at all. By distributing the system's intelligence we manage to make it very scalable. New data can be included just by adding it to the appropriate table in the Data sub-module and associating it with a rule in the Data Manipulation sub-module. New rules can be included by adding them in the Data Manipulation sub-module and associating them with some data. New user requirements can be included by adding decision making statements in the Web module.

FUTURE WORK

Our future work will aim at improving both techniques described in this paper. We believe that the client cache related technique can be improved much further, by using what we call time related Web page component caching. Current Web browser caching techniques implement full Web page caching schemes. These techniques are very much the same as the ones used by proxy servers. A local client cache can be seen, with some abstraction, as a local client proxy. In other words a local caching scheme implements a proxy server scheme in a much narrower sense. Cache and proxy schemas usually keep whole pages and not page components. A page component is a set of scripting language instructions that can be found between a certain HTML tag. If it was possible to cache page components and not the whole page it would be much easier to characterize these components as time related or dynamic. In such a case a server would know which components should be executed each time a certain page was requested by a certain client during that client's session. All the other components would only have to be executed once. In our opinion this would be a major breakthrough in the attempt to minimize redundant server execution and interaction with the client times.
The modular Web-database system that is presented in this paper can also be improved. It can be improved by further distribution of the system's intelligence and by introducing different database schemas for each sub-module that is described in this paper. An improvement in modularization can be achieved in the Web module in terms of differentiating between a client request sub-module and a decision-making sub-module. Different database schemas for each database sub-module on the other hand, would be very useful because they would help the process of the insertion of new data or new rules.

CONCLUSIONS

In this paper we have proposed a technique for improving performance of Web and Database Interaction and a technique that suggests how a WDI system can improve Web design. Both techniques are being tested and documented at the moment. We believe that both can accommodate a great deal of diverse applications in the future. The Web design model that is presented includes the database concept as one of the fundamental Web design components. This is going to be the future trend in our opinion because of the diverse (e.g. multimedia) information that a database can hold. The diversity of the Web page content is also getting larger in our days and because of this databases have become an integral component of Web design.

REFERENCES

ELECTRA ISPO project (1998); http://www.electra.eu.org/
Fedorov, A., Francis, B., Harrison, R., Homer, A., Murphy, S., Smith, R., Sussman, D. and S. Wood (1998); Professional Active Server Pages 2.0; Wrox Press (pp. 51-78, 217-319)
Hough, K. (1998); MSCD: SQL Server 6.5 Database Design (Study Guide); Sybex Network Press (pp. 31-70, 249-305, 573-594)
Kim, B.I. (1999); Web To Database Connectivity; available at: http://viu.eng.rpi.edu/dbconnect.html
Lazar, Z.P. and Holfelder P. (1997); Web Database Connectivity with Scripting Languages; Web Journal, Volume 2, Issue 2:”Scripting Languages: Automating the Web”; available at: http://www.ora.com/catalog/wj6/excerpt/index.html
Vaughn, W.R. (1998); Hitchhiker’s Guide to Visual Basic and SQL Server; Microsoft Press (pp. 3-92,139-152, 365-549)