A Web Based Mumps Virtual Machine

Kevin C. O’Kane, Computer Science Dept., University of Northern Iowa, Cedar Falls, IA 50614
Elizabeth E. McColligan, Datamedic Corp., 51 Sawyer Rd, Waltham, MA 02154

Abstract

This paper describes the development of a virtual machine environment to support decentralized, Mumps based medical record applications on the World Wide Web. This environment facilitates construction of efficient, fully functional, platform independent, multi-point medical record information systems that can be accessed any-where by low cost Web browsers to search, retrieve, download and analyze patient information. This environment also allows Mumps applications to operate in the heterogeneous Web system platform environment along with Web graphical, sound and video presentation services.

Introduction

In the distributed environment of the World Wide Web, platform independence for a programming language is the ability to run a program written in the language on any combination of target operating system and hard-ware. This is of critical importance if a host web server is to send programs to client browsers executing in un-known environments. Platform independence is one of the more important features of a language like Java. It is accomplished by developing a transportable virtual machine environment for the language for each target hardware and operating system so that programs behave exactly the same, regardless of the host on which they ultimately execute. The recent rapid growth in interest in languages like Java is due mainly to their ability to support programs that can reside on web servers and be downloaded and executed by any user browser. Entire office application suites are currently being re-coded to this model. For example, see:

http://officeforjava.corel.com

for an example of Java a based version of WordPerfect and Quattro Pro. These are Java programs that can be sent to the user’s browser for execution.

Many of the medically based Web systems recently proposed rely on the PERL or REXX scripting languages that, in turn, depend upon commercial data warehousing systems for data base services [1, 2, 3, 4, 5, 6, 7, 8]. As shown in Figure 1, applications based on this approach gather data and queries from browser users by means of HTML (HyperText Markup Language) based forms. The collected data is processed at the web server and the results are returned to the browser users in text or graphical format. All interaction is with server resident data and performed by programs operating under the Web server. The browser and user workstation generally function as little more than a sophisticated graphical terminal device. More recently, however, commercial systems have become available, generally only on large computers, in which the data base system itself is the web server.

A Typical Configuration

While the model described above is widely used, it suffers from several problems when applied to the healthcare domain. These are:

- Generally, it presumes that the Web server is situated on a relatively large system that supports DB2, SQL, Oracle or some other commercial data base package. The model is not suitable to small servers such as those that might be found in a small laboratory, physician office, radiology department and so
forth. The high processing overhead requirements of using third party data base engines, disk storage needs and

- Even in large systems, PERL, REXX and other intermediate script processor programs that rely upon SQL and DB2 for retrieval suffer from very high transaction overhead associated with the data base engine interfaces. These implementations are generally not scalable to high transaction volume environments. Current technology makes most existing applications that employ this approach suitable mainly as demonstrations. Even in those environments with limited transaction volume, this places limits on the scope of the information system.

- Script languages such as PERL, REXX and others derived from commercial data base engines, are more suited for system programming than medical record applications. They lack many features that are needed to accommodate the structure of the medical record and facilitate efficient retrieval.

- Although existing script languages can be piggy-backed onto commercial data base engines, all PERL, REXX and other script language based clinical applications must be newly written as no legacy systems exist to serve as a foundation. These applications are generally complicated to create and require personnel with a high level of clinical application development expertise.

- Existing script languages have minimal intrinsic facilities to support HTML document construction. Their structure is generally not compatible with program downloading thereby limiting display formatting capabilities that are essential to effective clinical applications.

- While commercial relational and network based data management systems efficiently support applications that require aggregation of patient data and the creation of cohorts, such as epidemiologic and administrative use (e.g., find all patients with test values in some specified date, time and value range), they are generally unsuited to patient care applications. Patient care applications require transaction oriented data bases that support individual patient data access. Different administrative and programming personnel make this model possible only in large centralized computing facilities. Patient record end uses demand different organizational approaches.

- None of the existing approaches lend themselves naturally to operating in a heterogeneous, multi-platform, distributed, networked model.

More recently, data base system vendors, including some versions of Mumps, have incorporated web servers directly into their products in order to eliminate PERL or REXX programs intermediaries. This approach generally locks the user into a large system, single language environment with little flexibility to access other services. It also limits the range of possible other programs that can be run concurrently for various needs. Ideally, a general purpose web server with programs that scale from small hardware platforms to large with no restrictions on the types of applications that can be developed, is most desirable.

This paper presents a Mumps virtual machine environment that is transportable to most popular desktop operating systems and has the following features:

- it is derived from Mumps and provides a full range of string handling routines and built-in functions;
- it is fully transportable to Unix, OS/2, Win95 and Win/NT environments;
- it provides full and easy access to user entered HTML form data;
- it has several extensions that aid in return of HTML document code to the server and client;
- it provides a B-tree based hierarchical and relational data base facility;
- it permits access to operating system facilities including other script processors and data base management systems;
- it is scalable from PC’s to mainframes;
- it is small, efficient, quickly loaded and executed, and it co-exists with other server based applications;
it can access all system facilities such as SQL, DB2 and Oracle;

it can both execute software sent from clients and it can download software applets for client execution;

and it can be run in standalone mode with or without a Web browser.

The Mumps language was developed in the mid-60's at the Massachusetts General Hospital [9]. It is a general purpose programming language with special facilities for hierarchical data base manipulation and enhanced string handling. It was widely used in biomedical settings and gained a following in several commercial environments as well. Both the Department of Defense CCHP and Veterans Administration DCHP systems, the two largest multi-hospital information systems in the world [10] are written in Mumps. A large number of interpreters for Mumps exist and it is available on many brands of computer systems at present. There are both ANSI and Department of Defense approved standards.

Unfortunately, as Mumps evolved over the years, what was originally a concise and efficient language for small machines evolved into a complex system with many operating system and hardware dependencies. In many cases, it is a large, multi-user based system platform with many idiosyncratic extensions. Additionally, most applications are character based, a definite weakness in today's graphical interface environment. Generally, these systems are not easily interfaced with Web servers except through inefficient PERL or REXX intermediated internal system data transfers as is the problem with SQL, DB2 and other data base engine approaches noted above.

System Description

The system described here is based on a compact Mumps shell processor for the 1986 standard of the Mumps language. It is derived from earlier work on Mumps interpreters [11,12,13,14,15,16,17]. The interpreter is written in C++ and is system independent. The resulting module generally is about 75,000 bytes of code as compared to over 500,000 bytes for many Perl processors. This difference affects the time required to load, initialize and process a web transaction.

A Mumps program consists of an ASCII file containing Mumps source code. When invoked by a web server, the interpreter then looks for any GET mode data passed by the CGI interface in the system environment variable QUERY_STRING. If any data is present in the environment variable QUERY_STRING, it is scanned, decoded, and decomposed into varname=value tokens that are then used to instantiate and initialize Mumps local variables. The varnames are derived from the NAME field of the HTML FORM tag and the values are derived from either the VALUE field or actual user entered text. The interpreter then executes the script and any standard output generated is captured by the web server for return to the originating browser. The interpreter may also be invoked in standalone mode operation without the CGI interface.

A user at a browser sends input to a program operating in the Web server cgi-bin interface through the HTML forms facility. In this mode of operation, the Web server begins the dialogue by sending to the Web browser an HTML encoded command stream that includes one or more instances of the <FORM> command. The <FORM> command causes the browser to display data input fields on the user's screen. These appear as data entry fields, check boxes, list boxes or radio buttons. When the browser user has completed entry of the data item or items or selection of one or more of the other multiple choice options, the Web browser encodes and sends the results to the Web server for processing. The web server stores the parameters from the browser in the environment variable QUERY_STRING and invokes the Mumps processor. The Mumps processor executes the appropriate Mumps program (the identity of which was one of the parameters) and returns a stream of text to the web server for transmission to the originating browser. The originating browser, receiving the results, displays them to the user.
Since Web servers are stateless, that is, they do not preserve information pertaining to the user state from one transaction to the next, special provisions must be made to permit sequential transactions. That is, data concerning the state of the system from the previous transaction is required to perform a subsequent operation. For example, if one transaction is used to identify a patient and a subsequent transaction retrieves lab data, the patient identification must be carried forward to the lab data retrieval module. A number of alternatives are possible. In our approach, we carry data forward by encoding in forms both visible and invisible pre-entered data fields. These fields are returned through the interface in the next transaction. This is a technique that is frequently employed in Web applications. Transaction security is another issue that must be considered. Currently, most servers support a password based user authentication and access control. Presently there are a number of robust security features including data encryption, with other under development. Encryption and other forms of security are handled at the browser and Web server levels and are consequently not discussed here.

Another important feature of this Mumps language processor is its ability to send and receive messages directly from/to other Web servers and browsers. This permits, for example, a central server to cause a remote server to execute programs with the results returned to the central server. Thus, a central server can update data at remote workstations dynamically by passing update data to the workstations and invoking the appropriate update routines.

**Distributed Multi-Host Virtual System**

Since this Mumps processor executes on most commonly used system platforms, it provides a virtual environment in which Mumps programs can execute, independent of the host operating system or hardware configuration as shown in Figure 2. A Mumps program written for one platform executes in exactly the same way on any other platform. This feature, which is also one of the primary advantages of Java, means that programs can be distributed by web servers to remote browser clients without regard to the configuration of the target system upon which they will ultimately execute.

Consequently a central server or, indeed, a collection of cooperating servers, can each dynamically exchange both programs and data without regard for the individual hardware and software configurations of the separate systems. While most web servers reside on larger central file servers, servers can also easily reside on individual, PC-based workstations, or for that matter, laptops, although their transaction processing power is certainly more limited than is the case with larger systems. In a fully distributed environment based on this Mumps system, data and programs can be downloaded to individual user workstations for local execution. This has considerable consequences for response times in high traffic environments. Of particular interest is the potential with frame based browsers to have multiple windows (frames) displaying the results from multiple, distributed, cooperating Mumps information hosts.

When a patient record is downloaded to a browser there is always the chance that an important element of the record may be updated or altered while the browser user is actively accessing the local copy of the record. In order to ensure that the browser based record is an accurate copy of the server based record, a table is maintained by the server indicating which records are at which browser. All downloaded records must be aged and become invalid after some set interval from the time of downloading and are deleted from the browser database. If a record is updated on the server, the server checks a table to determine which other systems, if any, have copies of the record. To those systems which have copies, a transaction request is sent to their web servers with instructions to update the local copy of the patient record to bring it into compliance with the server.
Conclusion

We believe that this environment facilitates construction of efficient, fully functional, platform independent, multi-point medical record information systems that can be accessed anywhere by low cost Web browsers to search, retrieve, download and analyze patient information.

Free copies of this Mumps processor along with documentation for Win95, Win/NT, OS/2, Linux and Sun Solaris operating systems are available at:

http://www.cs.uni.edu/~okane

A sample demonstration medical record system is also present along with links to related software. With these packages, it should be possible for a system designer to install in relatively short order a functional Intranet information system based on Mumps.

References


