Journal of Universal Computer Science, vol. 6, no. 6 (2000), 538-559
submitted: 16/11/99, accepted: 12/5/00, appeared: 28/6/00 © Springer Pub. Co.

Performance of RDBMS-WWW Interfaces under Heavy Workload[1]

Stathes Hadjiefthymiades
(Department of Informatics, University of Athens, Athens, Greece
shadj@di.uoa.gr)

Ioannis Varouxis
(Department of Informatics, University of Athens, Athens, Greece
varj@rocketmail.com)

Drakoulis Martakos
(Department of Informatics, University of Athens, Athens, Greece
martakos@di.uoa.gr)

Abstract: The WWW is currently considered as the most promising and rapidly evolving software platform for the deployment of applications in wide area networks as well as enterprise intranets. Interfacing legacy systems like RDBMS to the WWW has become a very important issue to the computing industry. We discuss the efficiency of RDBMS gateways throughout periods of increased workload. We present a client/server architecture aiming to diminish overheads encountered in conventional gateways. The performance gain is assessed through a series of measurements. Alternative architectures were subject to the same measurements to assess the performance achieved by technologies like ODBC, JDBC, Dynamic SQL, ISAPI, NSAPI and CORBA.

Key Words: WWW, RDBMS, gateway, SQL, CGI, Server API, Java, performance
Categories: H.3.3, H.3.4, H.4

1 Introduction

In the second half of the 90s, the WWW has evolved to the dominant software technology and broke several barriers. Despite the fact that it was initially intended for a WAN, the WWW, today, enjoys a unprecedented success and penetration even in corporate environments. Key players in the business software arena, like Oracle and Microsoft, recognise the success of this open platform and constantly adapt their products to it. Originally conceived as a tool for the co-operation between physicists, this network service is nowadays becoming synonymous with the Internet as it is used vastly by the majority of Internet users even for tasks like e-mail.

The universal acceptance of the WWW stimulated the need to provide access to the vast legacy of existing heterogeneous information[2]. Such information ranged from

[1] This work was supported by the General Secretariat of Research and Technology (GSRT).
[2] The significance of this issue triggered the organization, by W3C, of the workshop on Web Access to Legacy Data, in parallel to the 4th International WWW Conference, held in Boston, MA in December 1995.

proprietary representation formats to engineering and financial databases, which were, since the introduction of the WWW technology, accessed through specialised tools and individually developed applications. The vast majority of the various information sources were stored in RDBMS, a technology which enjoys wide acceptance in all kinds of applications and environments. The advent of the Web provided a unique opportunity for accessing such data repositories, through a common front-end interface, in an easy and inexpensive manner. The importance of the synergy of WWW and database technologies is also broadened by the constantly increasing management requirements for Web content ("database systems are often used as high-end Web servers, as webmasters with a million of pages of content invariably switch to a web site managed by database technology rather than using file system technology", extract from the Asilomar Report on Database Research [4]).

Initial research on the WWW-database framework did not address performance issues but since this area is becoming more and more important, relevant concern is beginning to grow. The main topic of this paper, namely the **study of the behaviour exhibited by WWW-enabled information systems under heavy workload**, spans a number of important issues, directly associated with the efficiency of service provision.

- Gateway specifications (e.g., CGI, ISAPI[3]) are very important due to their use for porting existing systems/applications to the WWW.
- The internal architecture of HTTP server software is also an important issue in such considerations. Older, single-process architectures are significantly slower than newer multi-threaded schemes. Pre-spawned server processes or threads also play an important role.
- Apart from the gateway specification used and the server architecture, the architecture of the interface towards the information repository (i.e., RDBMS) is also extremely important (even with the same specification quite different architectures may exhibit quite different behaviour in terms of performance).

In this paper we review the issues pertaining to the above listed topics and evaluate different technologies with the purpose of identifying the most efficient combinations.

The rest of the paper is structured as follows. In Section 2 we discuss popular gateway specifications like CGI and ISAPI. We also discuss how the internal architecture of a WWW server affects its performance. In Section 3, we initially study the architecture of database gateways. We discuss issues pertaining to database APIs. We then study different architectures for database gateways. Some schemes were subject to performance evaluation. The relevant results are also presented in this Section. We conclude this paper in Section 4.

[3] In this paper, Server APIs (e.g., ISAPI), will also be referred to as Gateway specifications despite the flexibility they offer for the modification of basic server functionality. Since we are mostly concerned with interfaces to RDBMSs, such feature of server APIs will not be considered.

2 Gateway Specifications and Server Architectures

In the following paragraphs, we provide a brief overview of major gateway specifications (e.g., CGI, NSAPI, ISAPI, WAI). We also discuss how the performance of servers is affected by their internal architecture.

2.1 Common Gateway Interface

The Common Gateway Interface (CGI) is a relatively simple interface mechanism for running external programs in the context of a WWW server in a platform independent way. The mechanism has been in use since 1993. The CGI specification is currently in the Internet Draft status [10]. Practically, CGI specifies a protocol for information exchange between the server and external programs as well as a method for their invocation by the server. Data are supplied to the program by the server through environment variables or the standard input file descriptor. CGI is a language independent specification. Owing to its simplicity, support for CGI is provided in almost all contemporary servers. A very important issue associated with the deployment of CGI is the execution strategy followed by the server. CGI-based programs run as short-lived processes separately to the server. As such, they are not capable of modifying basic server functionality or share resources with each other and the server. Additionally, they impose considerable resource waste and time overhead due to their one process/request scheme. During the evolution of WWW software, key commercial players like Netscape and Microsoft introduced their own, proprietary interfaces for extending basic server functionality. Such mechanisms are discussed below.

2.2 Netscape Server API

The NSAPI specification enables the development of server plug-ins (also called Service Application Functions or SAFs) in C or C++ that are loaded and invoked during different stages of the HTTP request processing. Since server plug-ins run within the Netscape server's process (such plug-ins are initialised and loaded when the server starts up), the plug-in functions can be invoked with little cost (no separate process needs to be spawned as in the case of CGI). Also, NSAPI exposes the server's internal procedure for processing a request. It is, therefore, feasible to develop a server plug-in that is invoked during any of the steps in this procedure. Such steps, in brief, are authorisation translation, name translation, path checking, object typing, respond to request[4] and transaction logging.

[4] This is the only step addressed by the CGI specification.

2.3 Web Application Interface

WAI [32] is one of the programming interfaces, provided in the latest Netscape servers, that allows the extension of their functionality. WAI is a CORBA-based programming interface that defines object interfaces to the HTTP request/response data as well as server information. WAI compliant applications can be developed in C, C++, or Java. WAI applications accept HTTP requests from clients, process them, and, lastly generate the appropriate responses. Server plug-ins may also be developed using WAI. Netscape claims that WAI outperforms CGI. Since WAI-compliant modules (or WAI services) are persistent, response times are reduced thus, improving performance. Additionally, WAI modules are multi-threaded so the creation of additional processes is unnecessary.

2.4 Internet Server API

ISAPI [29], [31] is an interface to WWW servers that allows the efficient extension of their basic functionality. ISAPI compliant modules run in Windows 9x/NT environments as dynamic link libraries (DLL). ISAPI DLLs can be loaded and called by a WWW server and provide similar functionality to CGI applications (such ISAPI DLLs are called extensions). The competitive advantage of ISAPI over CGI is that ISAPI code runs in the same address space as the server and has access to all its resources (thus, the danger of a server crash due to error-prone code is not negligible). Additionally, ISAPI extensions, due to their multi-threaded orientation, have lower overhead than CGI as they do not require the creation of additional processes upon reception of new requests and do not involve communications across process boundaries. Filters are another type of ISAPI modules. Filters are loaded once, upon server's boot, and invoked for each incoming request. ISAPI Filters allow the customisation of the flow of data within the server. ISAPI is used by several Web servers including Microsoft, Process Software, and Spyglass.

2.5 FastCGI

FastCGI [6] is basically an effort to provide a "new implementation of CGI" [7] that enjoys the portability of its predecessor while overcoming its performance handicap. FastCGI processes are persistent in the sense that after dispatching some request, they remain memory resident (and, do not exit, as conventional CGI) awaiting for another request to arrive. Instead of using environment variables, standard input and output, FastCGI communicates with the server process through a full duplex Socket connection. Basing the interface on Sockets allows the execution of the gateway instance on a machine different from that of the WWW server. The information transferred over this full-duplex connection is identical to the one exchanged in the CGI case. Thus, migration from classical CGI programs to FastCGI is fairly simple. FastCGI supports the language independence of its predecessor. FastCGI applications can be programmed either single-threaded or multi-threaded. The FastCGI framework supports a performance enhancement technique called “session affinity”. Session affinity allows the server to route incoming requests to specific memory resident copies of FastCGI processes based on information conveyed within requests (e.g., username/password of the authentication scheme). The performance benefit comes from the caching that applications are allowed to perform on user-related data. A side effect of this technique is that it facilitates session management in the WWW environment.

2.6 Servlet Java API

Servlets are protocol- and platform-independent server side components written in Java. Servlets execute within Java based web servers (e.g., Java Web Server) or within external “Servlet engines” interfaced to other types of web servers like Netscape and Apache (such technique will be discussed below). They are to the server side what applets are to the client side. Servlets can be used in many ways but generally regarded as a replacement to CGI. Servlets allow the realisation of 3-tier schemes where access to a legacy database or another system is accomplished through some technology like Java Database Connectivity (JDBC) or Internet Inter-ORB Protocol (IIOP). Unlike CGI, the Servlet code stays resident after the dispatch of an incoming request. To handle simultaneous requests new threads are spawned instead of processes.

2.7 Comparison between CGI and Server APIs

Gateway specifications are compared in terms of characteristics (qualitative) or the performance (quantitative) they can achieve. A detailed comparative analysis of the characteristics of CGI and server APIs can be found in [12] and [37]. Specifically, CGI is the most widely deployed mechanism for integrating WWW servers with legacy systems. However, its design does not match the performance requirements of contemporary applications: CGI applications do not run within the server process. In addition to the performance overhead (new process per request), this implies that CGI applications can't modify the behaviour of the server's internal operations, such as logging and authorisation (see Section 2.2). Finally, CGI is viewed as a security issue, due to its connection to a user-level shell.

Server APIs can be considered as an efficient alternative to CGI. This is mainly attributed to the fact that server APIs entail a considerable performance increase and load decrease as gateway applications run in or as part of the server processes (practically the invocation of the gateway module is equivalent to a regular function call[5]) instead of starting a completely new process for each new request, as the CGI specification dictates. Furthermore, through the APIs, the operation of the server process can be customised to the individual needs of each site. The disadvantages of the API solution include the limited portability of the gateway code which is attributed to the absence of standardisation (completely different syntax and command sets) and strong dependence to internal server architecture. The choice for the programming language

[5] Differences arise from the start-up strategies followed: some schemes involve that gateway instances/modules are pre-loaded to the server while others follow the on-demand invocation.

in API configurations is extremely restricted if compared to CGI (C or C++ Vs C, C++, Perl, Tcl/Tk, Rexx, Python and a wide range of other languages). As API-based programs are allowed to modify the basic functionality offered by the web server, there is always the concern of buggy code that may lead to crashes.

The two scenarios involve quite different resource requirements (e.g., memory) as discussed in [37]. In the CGI case, the resource needs are proportional to the number of clients that are simultaneously served. In the API case, resource needs are substantially lower due to the function-call like implementation of gateways and multi-threaded server architecture.

In terms of performance, many evaluation reports have been published during the past years. Such reports clearly show the advantages of server APIs over CGI or other similar mechanisms but also discuss performance differences between various commercial products.

The qualitative and quantitative comparison of the discussed schemes is shown in Table 1. Performance considerations are based on a series of published reports like [8], [9], [14], [19].

	CGI	**ISAPI**	**NSAPI**	**WAI**	**Servlet API**	**FastCGI**
Performance	-	+ +	+ +	+	+	+
Language Independence	+ +	-	-	+ +	-	+ +
Portability	+ +	-	-	+	+	+ +
Popularity	+ +	-	-	-	+[6]	+
Low level server programming	-	+ +	+ +	+	+ +	-
+ + : Good, + : Medium, - : Poor						

Table 1: Qualitative/quantitative comparison

2.8 Web Server Architectures

The architecture of HTTP demons (or WWW servers) has been a very important issue since the advent of the WWW. The concern of the WWW community about the implications of WWW server architecture on performance is clear in works like [28] and [24]. The very first servers were designed to fork a new process for each incoming request. Newer architectures adopt the, so-called, pre-forking (or pool of processes) approach. The pre-forking approach involves the provisional generation, by the master process, of a set of slave processes. Such processes are waiting idle until an HTTP request reaches the master process. Then, the master process accepts the connection and passes the file descriptor to one of the pre-spawned clients. If the number of simultaneous requests at the server exceeds the number of pre-spawned clients, then the master process starts forking new instances. Measurements reported in [28] show that this technique practically doubles the performance of the WWW server.

[6] Servlet Engines are also taken into account.

Newer server architectures were designed in the more efficient, multi-threaded paradigm. Notable examples are Netscape's servers. In those architectures, upon WWW subsystem's boot, only one server process is spawned. Within this server process the pool of available instances principle is still followed but on the thread level. In other words, a number of threads are pre-spawned and left idle, awaiting incoming requests to serve. If the threads in the thread pool are all in use, the server can create additional threads within the same process/address space to handle pending connections. Servers are optimised on the thread model supported by the underlying OS (e.g. in the HP-UX case where not native threading is supported a user-level package is provided).

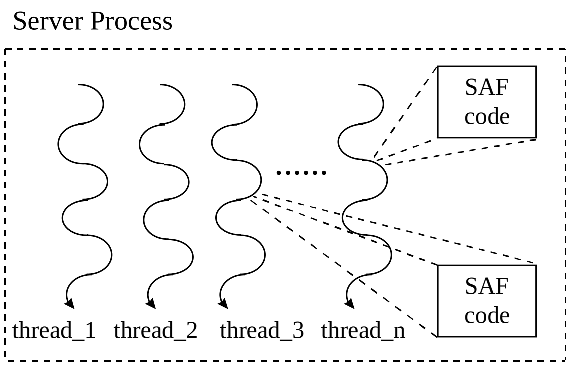

Figure 1: Multi-threaded Netscape Server architecture

As discussed in Section 2.2, gateways can be built in Netscape servers using the NSAPI specification. Taking into account the server's multi-threaded architecture, the performance benefit is two-fold: (a) SAF/plug-in code is pre-loaded in the memory space of the server process and (b) the server spawns autonomous threads instead of forking processes for each incoming request (Figure 1). Microsoft servers exhibit quite similar behaviour.

3 Architectures of RDBMS Gateways

One of the most important applications of the WWW platform refers to the integration of database management systems and RDBMS in particular. Many issues are associated with this particular type of WWW applications, namely the generic architecture, the stateful operation and performance, which is of prime concern in this paper. The first WWW interfaces for relational management systems appeared in the '93-94 period. The first products begun to come into sight on '95 with WebDBC [33] from Nomad (later StormCloud), Cold Fusion [1] from Allaire and dbWeb [26] from Aspect Software Engineering. The importance of such middleware triggered a very rapid pace in the development of this kind of software. Such interfacing tools have become crucial and indispensable components to the Internet as well as enterprise intranets.

The first generation of WWW-RDBMS interfacing products was mainly intended for the Windows NT platform and capitalised on ODBC to gain access to databases.

Throughout the deployment of WWW gateways for RDBMSs it can be easily observed that, during users' queries for information, a considerable amount of time was spent for the establishment of connections towards the relational system. **Irrespectively of the efficiency of the adopted gateway mechanism (i.e., the "slow" CGI Vs a "fast" server API), establishing a new connection (per request) to the relational system is a time- and resource - consuming task which should be limited and, if possible, avoided.**

In [15], the scheme of permanent connections towards the database management system was adopted [37]. Permanent connections are established by one or more demon processes that reside within the serving host and execute queries on the behalf of specific clients (and gateway instances). Demons, prevent the inefficient establishment of a large number of database connections and the relevant resource waste; the associated cost is incurred by the demon processes (prior to actual hit dispatch - query execution) and not by the gateway instances (e.g., CGI script, ISAPI thread) upon hit dispatch. Thus, no additional time overhead is perceived by the interacting user in his queries. The discussed scheme is shown in Figure 2.

3.1 Generic DBMS Interfaces

A very important issue in the demon-based architecture shown in Figure 2 is the interface towards the DBMS (interface D) as well as the communication mechanism between the gateway instance (interface C in Figure 2). The demon process should be able to dispatch any kind of SQL statements irrespective of database, table and field structures. This requirement discourages the adoption of a static interface technique like the embedding of SQL statements in typical 3GLs (i.e., the Embedded SQL API - ISO SQL-92 standard). Instead, a generic interface towards the database should be used.

Contemporary RDBMS offer, as part of the Embedded SQL framework, the Dynamic SQL capability [20], [30], [35] which allows the execution of any type of statement without prior knowledge of the relevant database schema, table and field structures (unlike Static SQL). Dynamic SQL statements can be built at runtime and placed in a string host variable. Subsequently, they are sent to the RDBMS for processing. As the RDBMS needs to generate an access plan at runtime for dynamic SQL statements, dynamic SQL is slower than its static counterpart. This last statement deserves some more discussion: if a generic interface towards a DBMS is pursued, then the execution efficiency of the system could be undermined. Embedded SQL scripts with hard-coded statements execute faster than CGI scripts with dynamic SQL capabilities. The real benefit, in this architecture, comes from the de-coupling of the dynamic SQL part (database demon) from the actual gateway instance (Figure 2).

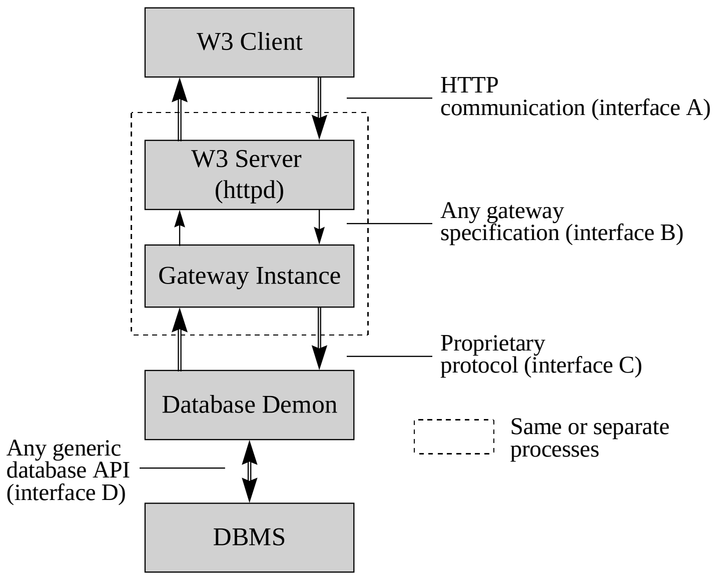
W3 Client
HTTP communication (interface A)
W3 Server (httpd)
Any gateway specification (interface B)
Gateway Instance
Proprietary protocol (interface C)
Database Demon
Same or separate processes
Any generic database API (interface D)
DBMS

Figure 2: Optimised WWW-DBMS interface

A second option for generic access to a DBMS is the SQL Call-Level Interface (CLI). The SQL CLI was originally defined by the SQL Access Group (SAG) to provide a unified standard for remote data access. The CLI requires the use of intelligent database drivers that accept a call and translate it into the native database server's access language. The CLI is used by front-end tools to access the RDBMS; the latter should incorporate the appropriate driver. The CLI requires a driver for every database to which it connects. Each driver must be written for a specific server using the server's access methods and network transport stack. The SAG API is based on Dynamic SQL (statements need to be Prepared[7] - Executed). On 1994, the SAG CLI became X/Open specification (currently it is also referred to as X/Open CLI [27]) and later on an ISO international standard (ISO 9075-3 [21]). Practically, the X/Open CLI is a SQL wrapper (a procedural interface to the language); a library of DBMS functions - based on SQL - which can be invoked by an application.

Microsoft's Open DataBase Connectivity (ODBC) API is based on the X/Open CLI. The 1.0 version of the specification and the relevant Software Development Kit (SDK), launched by Microsoft in 1992, have been criticised for poor performance. Initially, ODBC was confined to the Windows platform, but was later ported to platforms like Solaris. The ODBC 2.0 (1994) has been considerably improved over its predecessor. 32-bit support contributed to the efficiency of the new generation of ODBC drivers. In 1996, Microsoft announced ODBC 3.0. Nowadays, most database vendors support ODBC in addition to their native SQL APIs.

[7] It is requested by the RDBMS to parse, validate, and optimise the involved statement and, subsequently, generate an execution plan for it.

Since the advent of the Java programming language, a new SQL CLI has emerged. It is named JDBC (Java DataBase Connectivity) and was jointly developed by Javasoft, Sybase, Informix, IBM and other vendors. JDBC is a portable, object-oriented CLI, written entirely in Java but very similar to ODBC [36]. It allows the development of DBMS independent Java code which is, at the same time, independent of the executing platform. JDBC's architecture, similarly to ODBC, introduces a driver manager (JDBC driver manager) for controlling individual DBMS drivers. Applications share a common interface with the driver manager. A classification of JDBC drivers suggests that they are either direct or ODBC-bridged. Specifically, there are four types of JDBC drivers [23]:

- Type 1 refers to the ODBC-bridged architecture and involves the introduction of a translation interface between the JDBC and the ODBC driver. ODBC binary code, and the required database client code must be present in the communicating party.
- Type 2 drivers are based on the native protocols of individual DBMS (i.e., vendor-specific) and were developed using both Java and native code (Java methods invoke C or C++ functions provided by the database vendor).
- Type 3 drivers are exclusively Java based. They use a vendor-neutral protocol to transmit (over TCP/IP sockets) SQL statements to the DBMS thus, necessitating the presence of a conversion interface (middleware) on the side of the DBMS.
- Type 4 drivers are also exclusively based on Java (pure Java driver) but, in contrast to Type 3, use a DBMS specific protocol (native) to deliver SQL statements to the DBMS.

In the some of the experiments documented in this paper (Section 3.3) both Type 2 and Type 4 JDBC drivers have been employed.

3.2 Protocols and Inter-process Communication Mechanisms

Another very important issue in the architecture shown in Figure 2 is the C interface (i.e., the interface between gateway instances and the database demon). The definition of interface C involves the adoption of a protocol between the two co-operating entities as well as the selection of the proper IPC (Inter-Process Communication) mechanism for its implementation [15].

In [15] a simplistic, request/response, client/server protocol for the realisation of the interface was proposed. The gateway instance transmits to the database demon a ClientRequest message and the database demon responds with a ServerResponse. The ClientRequest message indicates the database to be accessed, the SQL statement to be executed, an identifier of the transmitting entity/instance as well as the layout of the anticipated results (results are returned merged with HTML tags). The Backus-Naur Form (BNF) of ClientRequest is:

```
ClientRequest       = DatabaseName SQLStatement
                      [ClientIdentifier] ResultsLayout
DatabaseName        = *OCTET
SQLStatement        = *OCTET
ClientIdentifier    = *DIGIT ; UNIX PID
ResultsLayout       = "TABLE" | "PRE" | "OPTION"
```

ClientIdentifier is the process/thread identifier of the gateway instance that generated the request. This field is optional, depending on the IPC mechanism used and could be avoided in a connection-oriented communication (e.g., Sockets). *OCTET denotes a sequence of printable characters and thus, represents a text field. Results are communicated back to the client processes by means of the ServerResponse message. The BNF of ServerResponse is:

```
ServerResponse   = ResponseFragment ContinueFlow
ResponseFragmet  = *OCTET
ContinueFlow     = "YES" | "NO"
```

The ResponseFragment (text) field contains the actual information that was retrieved by the demon process from the designated database. As mentioned, such information is returned to the client, embedded within HTML code. The type of tags used is the one specified in the ResultsLayout field of ClientRequest. The ContinueFlow field is used for optimising, in certain IPC scenarios (e.g., Message Queues), the transmission of results back to the gateway instance.

The ClientRequest-ServerResponse protocol is very simplistic, and shifts processing from the gateway instance (client) to the database demon (a “thin” client scheme). More advanced protocols (i.e., RDA/DRDA) may be used over the C interface to allow more complicated processing at the side of the gateway instance. The widely established XML standard [39] also provides a flexible platform for the exchange of structured, relational data [40], [41], [42]. Should the XML be used for the implementation of the ClientRequest-ServerResponse protocol, additional processing is required on the side of the gateway instance for parsing [39] the transmitted ResponseFragments. XML can be used in Microsoft server architectures for the flow of database information from MTS (Microsoft Transaction Server) objects to Active Server Pages (ASP) where XSL transformations are invoked for generation of the appropriate HTML code.

Protocols pertaining to the C interface are deployed using either some IPC mechanism like Message Queues [15], [17] or Sockets [18] or some kind of middleware like CORBA/IIOP. Message Queues [38] are a quite efficient, message oriented, IPC mechanism that allows the simultaneous realisation of more than one dialogues (i.e., various messages, addressed to different entities, can be multiplexed in one queue). BSD Sockets are ideal for implementation scenarios where the web server (and, consequently, the gateway instances) and the database demon execute on different hosts. The advent of the WinSock library for the Microsoft Windows environments rendered Sockets a universal, platform independent IPC scheme. Message Queues are faster than Sockets but restrict the communication in a single host since they are maintained at the kernel. In some recent tests, which are also presented in this paper, for the realisation of a scheme similar to the one shown in Figure 2, we have employed Named Pipes in the Windows NT environment. A Named Pipe is a one-way or two-way communication mechanism between a server process and one or more client processes executing on the same or different nodes (networked IPC).

A type of middleware, used extensively nowadays in many application domains, is CORBA [34]. CORBA simplifies the development of distributed applications with components that collaborate reliably, transparently and in a scaleable way. The efficiency of CORBA for building interfaces between Java applications is discussed in [36]. It is reported that CORBA performs similarly (and, in some cases better) to Socket based implementations while, only buffering entails a substantial improvement in Socket communications. Another comparison between several implementations of CORBA (e.g., Orbix, ORBeline) and other types of middleware like Sockets can be found in [13]. In particular, low level implementations such as Socket-based C modules and C++ wrappers for Sockets significantly outperformed their CORBA or RPC competitors. Differences in performance ranged from 20 to 70% depending on the data types transferred through the middleware (transmission of structures with binary fields has proved considerably “heavier” than scalar data types).

3.3 Experiments and Measurements

In this section, we present two series of experiments which allow the quantification of the time overheads imposed by conventional gateway architectures and the benefits that can be obtained by evolved schemes (such as the one shown in Figure 2).

Firstly, we examined the behaviour of a web server setup encompassing a Netscape FastTrack server and Informix Online Dynamic Server (ver.7.2), both running on a SUN Ultra 30 workstation (processor: SUN Ultra 250 MHz, OS: Solaris 2.6) with 256 MB of RAM. In this testing platform we have evaluated the demon-based architecture of Figure 2 and the ClientRequest/ServerResponse protocol of Section 3.2, using, on the B interface, the CGI and NSAPI specifications (all tested scenarios are shown in Figure 3) [17]. The IPC mechanism that we have adopted was Message Queues. Quite similar experiments were also performed with BSD Sockets [18] but not reported in this section. Both types of gateway instances (i.e., CGI or NSAPI) as well as the server demon were programmed in the C language. The server demon, for the D interface, used the Dynamic SQL option of Embedded SQL (Informix E/SQL). Only one database demon existed in the setup. Its internal operation is shown in Figure 4, by means of a flowchart. It is obvious, from Figure 4, that the database demon operates in a generic and iterative way, accessing any of the databases handled by the DBMS and executing any kind of SQL statement. If the, until recently, used DBMS connection (e.g., to a database or specific user account) is the same with the connection needed by the current gateway request then, that connection is being used. The JDBC 2.0 specification [43] adopts a very similar technique named **Connection Pooling**. When some application closes a database connection, that connection is recycled rather than being released. According to Javasoft, “**reusing connections can improve performance dramatically by cutting down the number of new connections that need to be created**”.

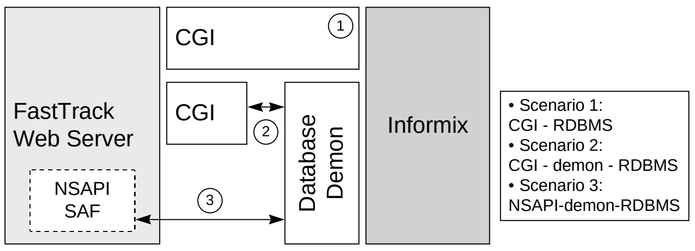
FastTrack Web Server
NSAPI SAF
CGI
1
CGI
2
3
Database Demon
Informix
• Scenario 1: CGI - RDBMS
• Scenario 2: CGI - demon - RDBMS
• Scenario 3: NSAPI-demon-RDBMS

Figure 3: Database access scenarios (1)

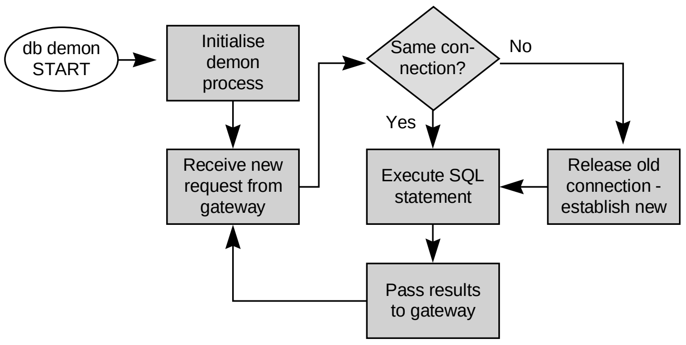

Figure 4: Flowchart for Database Demon operation

As shown in Figure 3, we compared the conventional (monolithic) CGI-based Informix gateway (C and Embedded SQL - Static SQL option) against the CGI $\leftrightarrow$ Database Demon $\leftrightarrow$ Informix scheme (Scenario 2) and the NSAPI SAF $\leftrightarrow$ Database Demon $\leftrightarrow$ Informix combined architecture (Scenario 3). In all three cases, the designated database access involved the execution of a complicated query over a sufficiently populated Informix table (around 50,000 rows). The table contained the access log accumulated in a web server over a period of six months. The layout of the table followed the Common Log Format found in all web servers and the row size was 196 bytes. The executed query was the following: Select the IP address and total size of transmitted bytes (grouping by the IP address) from the access log table where the HTTP status code equals 200 (i.e., Document follows). The size of the HTML page produced was 5.605 KB in all scenarios (a realistic page size, taking into account the published WWW statistics [3], [5]). The tuples extracted by the database were embedded in an HTML table (ResultsLayout = "TABLE").

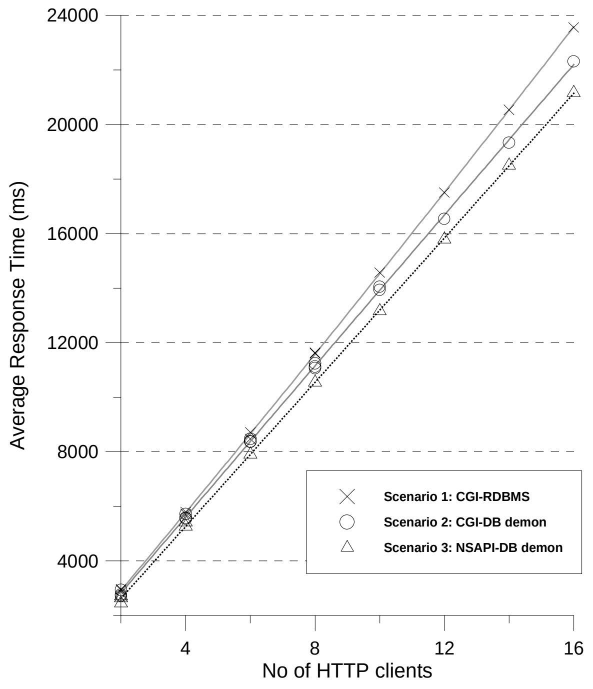

Figure 5: Response Time Vs Number of Clients

The above-described experiments were realised by means of an HTTP pinger, a simple form of benchmarking software. The pinger program was configured to emulate the traffic caused by up to 16 HTTP clients. In each workload level (i.e., number of simultaneous HTTP clients), 100 repetitions of the same request were directed, by the same thread of the benchmark software, to the WWW server. The recorded statistics included:

- Connect Time (ms): the time required for establishing a connection to the server.
- Response Time (ms): the time required to complete the data transfer once the connection has been established.
- Connect rate (connections/sec): the average sustained throughput of the server.
- Total duration (sec): total duration of the experiment.

The pinger program executed on a MS-Windows NT Server (version 4) hosted by a Pentium II 300 MHz machine with 256 MB of RAM and a PCI Ethernet adapter. Both machines (i.e., the pinger workstation as well as the web/database server) were interconnected by a 10Mbps LAN and were isolated by any other computer to avoid additional traffic that could endanger the reliability of the experiments. From the gathered statistics, we plot, in Figure 5, the Average Response Time per request. The scatter plot of Figure 5 is also enriched with polynomial fits.

Figure 5 shows that the CGI ↔ demon architecture (Scenario 2) performs systematically better than the monolithic CGI gateway (Scenario 1) and worst than the NSAPI ↔ Demon configuration (Scenario 3), irrespective of the number of HTTP clients (i.e., threads of the pinger program). The performance gap of the three solutions increases proportionally to the number of clients. Figure 5 also suggests that for relatively small/medium workload (i.e., 16 simultaneous users) the serialisation of ClientRequests in Scenarios 2 and 3 (i.e., each ClientRequest incurs a queuing delay due to the iterative nature of the single database demon) does not undermine the performance of the technical option.

A number of additional tests were performed in order to cover even more specifications and gateway architectures. Such tests were performed in the Oracle RDBMS (ver. 7.3.4) running on a Windows NT Server operating system (version 4). The web server setup included Microsoft's Internet Information Server (IIS) as well as Netscape's Fasttrack Server (not operating simultaneously). Both the web servers and RDBMS were hosted by a Pentium 133 HP machine with 64MB of RAM. In this setup we employed, on the B interface (Figure 2), the CGI, NSAPI, ISAPI and Servlet specifications, already discussed in previous paragraphs. On the D interface we made use of Embedded SQL (both Static and Dynamic options), ODBC and JDBC. Apart from conventional, monolithic solutions, we have also evaluated the enhanced, demon - based architecture of Figure 2. All the access scenarios that were subject to evaluation are shown in Figure 6.

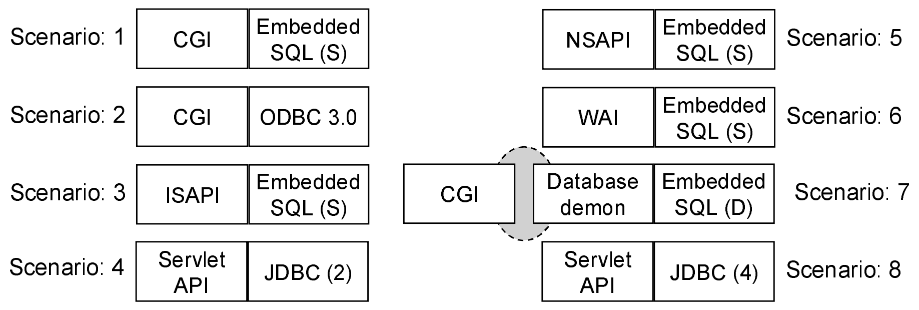

Figure 6: Database access scenarios (2)

The IPC mechanism that we adopted for the demon - based architecture (Scenario 7 - Figure 6) was Named Pipes. All modules were programmed in the C language (MS-Visual C). Similarly to the previous set of experiments, only one database demon (Dynamic SQL) existed in this setup. The flowchart of its internal operation is identical to the one provided in Figure 4. The database schema as well as the executed query were also identical to the previous series of experiments.

In this second family of experiments we employed the same HTTP pinger application with the previously discussed trials. It executed on the same workstation hosting the two web servers as well as the RDBMS. The pinger was configured to emulate the traffic caused by a single HTTP user. Each experiment consisted of 10 repetitions of the same request transmitted towards the web server over the TCP loop-back interface. As in the previous case, Connect Time, Response Time, Connect Rate and Total Duration were the statistics recorded. Apart from those statistics, the breakdown of the execution time of each gateway instance was also logged. This was accomplished by enriching the code of gateway instances with invocations of the C `clock()` function which returns the CPU time consumed by the calling process. Thus, we were able to quantify the time needed for establishing a connection to the RDBMS (to be referred to as T_{con}) as well as the time needed to retrieve the query results (to be referred to as T_{ret}). Such logging of CPU times was performed in scenarios:

- 1 (CGI/Embedded SQL),
- 2 (CGI/ODBC),
- 4 (Servlet/JDBC – Type 2) [8],
- 7 (CGI ↔ Database Demon/Dynamic SQL), and,
- 8 (Servlet/JDBC – Type 4)

We restricted the time breakdown logging in those scenarios since they involve different database access technologies. Figure 7 plots the Average Response Time for each scenario.

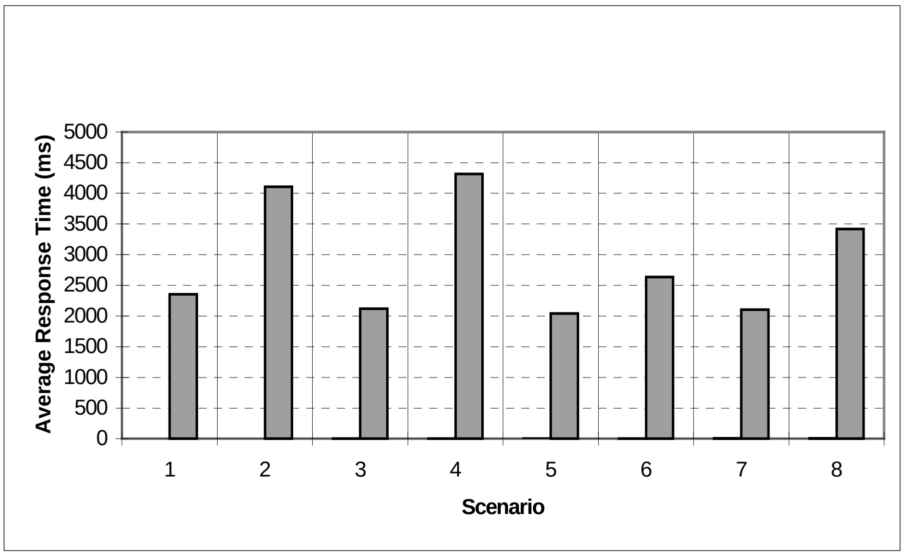

Figure 7: Response Times for Scenarios 1-8

Figure 7 shows that the CGI ↔ Demon architecture accomplishes quite similar times to those of the NSAPI and ISAPI gateways. In the multi-threaded gateway specifications (i.e., NSAPI, ISAPI and WAI), connections towards the RDBMS are established by the executing thread (hence, the associated cost is taken into account). Connections by multi-threaded applications are only possible if mechanisms like Oracle's Runtime Contexts [35] or Informix's dormant connections [20] are used. Other multi-threaded configurations are also feasible: a database connection (and the associated runtime context) could be pre-established (e.g., by the DLL or the WAI application) and shared among the various threads, but such configurations necessitate the use of synchronisation objects like mutexes. In such scenarios a better performance is

[8] Gefion's WAICoolRunner was used as a Servlet engine in this scenario.

achieved at the expense of the generality of the gateway instance (i.e., only the initially opened connection may be re-used).

In Scenario 2, ODBC Connection Pooling was performed by the ODBC Driver Manager (ODBC 3.0) thus, reducing the time overhead associated with connection establishments after the initial request. In Scenarios 4 and 8, the JDBC drivers shipped with Oracle 7.3.4 were used. Specifically, we employed Type 2 and Type 4 drivers (see Section 3.1) in Scenarios 4 and 8 respectively. In Figure 8 we show where the factors T_{con} and T_{ret} range.

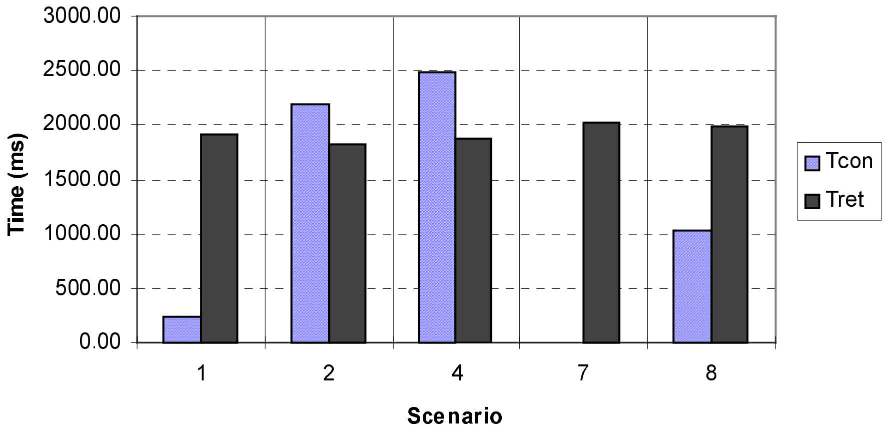

Figure 8: Time breakdown for Scenarios 1, 2, 4, 7 and 8

As shown in Figure 8, in the ODBC and JDBC/Type2 scenarios (i.e., Scenarios 2 and 4) a very important percentage of the execution time of the gateway is consumed for the establishment of connections towards the DBMS. The Embedded SQL scenario (i.e., Scenario 1) achieves the lowest T_{con}. The highest T_{ret} is incurred in Scenario 7 where query execution is fully dynamic (see Section 3.1 for the Dynamic SQL option of Embedded SQL). A marginally lower T_{ret} is incurred in Scenario 8. T_{con} is not plotted in Figure 8 for Scenario 7, as this cost is only incurred once by the database demon.

3.4 State Management Issues

In general, stateful web applications impose considerable additional overhead. Extra information is carried across the network to help identify individual sessions and logically correlate user interactions. The IETF RFC for the Cookies mechanism [25] specifies additional fields in the HTTP headers to accommodate state information. Complicated applications, though, may require a large volume of Cookie information to be transmitted to and from the browser. At the server's side, additional processing - handling is required in order to perform state management. In [16] a framework for the adaptation of stateful applications to the stateless Web has been proposed. This framework involved the placement of an extra software layer in the browser-database chain. Such additional, server-side, tier would associate incoming Cookie values with the appropriate threads of a multi-threaded database application, pass execution information, relay results and assign new Cookie values to the browser in order to preserve a correct sequence of operations. An additional entity responsible for handling state information (conveyed as extra information in URLs) is also required in the architecture presented in [22]. Considerations concerning the overheads introduced by state management techniques are also discussed in [37]. Another technique that facilitates state management is the already discussed “session affinity” scheme introduced in the FastCGI platform.

The need for state management imposes enhancements to the protocol of the C interface (not included in the lightweight ClientRequest - ServerResponse protocol). The protocol changes largely depend on the requirements for state management introduced by the application [16]. Additionally, layers/tiers placed between the legacy application or the DBMS and the WWW server considerably increase response times (Figure 9).

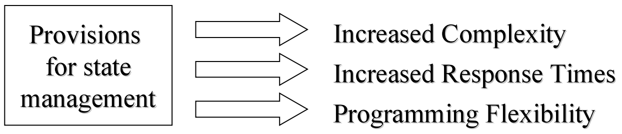

Figure 9: Implications of state management

3.5 Persistent Vs Non-persistent Connections to Databases

The database connection persistence problem is discussed in [37]. A conventional scheme, with monolithic CGIs establishing connections directly towards the RDBMS, could exhaust the available licenses due to the sporadic nature of WWW requests [3], [11]. Additionally, as previously discussed, a substantial time overhead is to be incurred in this scheme. Maintaining a database connection per active session (e.g., through a server API or FastCGI) may not be a sound strategy as the maximum licenses limit may also be easily reached while extensive periods of client inactivity cause a waste in resources. In these two scenarios, uncontrolled connection establishment with the RDBMS is, surely, a problem. Persistent database connections may, however, be beneficial for response times, as state management is simplified [16]. A demon based architecture, on the other hand, restricts the number of connections simultaneously established towards the RDBMS (controlled rate of connection establishment). Multiple demons may be established to serve incoming requests in a more efficient way (so as to reduce potential queuing delays caused by the availability of a single dispatching entity). In such scenario a regulating entity will be needed to route requests to idle demons (similarly to the load balancing techniques discussed in [2]).

3.6 Template - based Middleware

Software tools intended for building database-powered WWW sites usually follow a template approach. A generic engine interprets pre-programmed templates and performs database interactions as needed (run-time binding of re-usable code, eg. some ODBC front-end interface, with a template). Templates practically are the combination of HTML with a proprietary tag set (dealing with query specification, transactions, result set handling, flow control, etc.). An indicative example of a template-based middleware is Allaire's Cold Fusion with its Cold Fusion Markup Language (CFML). Surely, the interpreter-like approach, dictated by the adoption of templates, causes a slow-down in the dispatch of queries. On the other hand, such tools are extremely efficient programming tools that drastically reduced the time required to develop database gateways in the old-fashioned way using techniques such as Embedded SQL.

4 Conclusions - Future Work

Database connectivity is surely one of the most important issues in the constantly progressing area of WWW software. More and more organisations are using the WWW platform for exposing their legacy data to the Internet. On the other hand, the amazing growth of WWW content forces the adoption of technologies like RDBMS for the systematic storage and retrieval of such information. Efficiency in the mechanisms intended for bridging the WWW and RDBMS is a very crucial topic. Its importance stems from the stateless character of the WWW computing paradigm that necessitates a high frequency of short-lived connections towards the database systems. In this paper, we have addressed a series of issues associated with the considered area of WWW technology. We have evaluated different schemes for database gateways (involving different gateway specifications and different types of database middleware). Although aspects like generality, compliance to standards, state management and portability are extremely important their pursuit may compromise the performance of the database gateway. The accumulated experience from the use and development of database gateways over the last 5-6 years suggests the use of architectures like the database demon scheme, which try to meet all the above mentioned requirements to a certain extend but also exhibit performance close to server APIs.

In the near future it is our intention to assess the performance of the considered architecture in configurations involving more than one database demons. An appropriate load balancing software module is currently under design / development. We are also planning to develop a variation of the presented architecture using the CORBA middleware technology and taking provisions for state management.

References

[1] “COLD FUSION User's Guide Ver. 1.5”, Allaire Corp. 1996.

[2] V. Cardellini, M. Colajanni, and P.S. Yu, “Dynamic Load Balancing on Web-Server Systems,” IEEE Internet Computing, Vol. 3, No. 3, 1999.

[3] P. Barford, and M. Crovella, “Generating Representative Web Workloads for Network and Server Performance Evaluation”, proceedings of ACM SIG-METRICS, International Conference on Measurement and Modeling of Computer Systems, July, 1998.

[4] P. Bernstein et al., “The Asilomar Report on Database Research”, ACM SIG-MOD Record, Vol. 27, No. 4, Dec. 1998.

[5] T. Bray, “Measuring the Web”, Computer Networks and ISDN Systems, Vol. 28, No. 7-11, 1996.

[6] M. Brown, “FastCGI Specification”, Open Market Inc., http://fastcgi.idle.com/kit/doc/fcgi-spec.html, April 1996.

[7] M. Brown, “FastCGI: A High Performance Gateway Interface”, Position paper for the workshop “Programming the Web - a search for APIs”, 5th International WWW Conference, Paris, France, 1996.

[8] “Performance Benchmark Tests of Microsoft and NetScape Web Servers”, Haynes & Company - Shiloh Consulting, http://www.tedhaynes.com/haynes1/infoserv/haynes1.htm, February 1996.

[9] P.I. Chang, “Inside the Java Web Server: An Overview of Java Web Server 1.0, Java Servlets, and the JavaServer Architecture”, http://java.sun.com/features/1997/aug/jws1.htm, 1997.

[10] K. Coar, and D. Robinson, “The WWW Common Gateway Interface - Version 1.2”, Internet Draft, February, 1998.

[11] M. Crovella, M. Taqqu, and A. Bestavros, “Heavy-Tailed Probability Distributions in the World Wide Web”, in “A Practical Guide to Heavy Tails - Statistical Techniques and Applications”, R. Adler, R. Feldman, and M. Taqqu (ed.), BIRKHAUSER, 1998.

[12] P. Everitt, “The ILU Requested: Object Services in HTTP Servers”, W3C Informational Draft, March, 1996.

[13] A. Gokhale and D. C. Schmidt, “Measuring the Performance of Communication Middleware on High Speed Networks”, proc. ACM SIGCOMM Conference, 1996.

[14] C. Karish, and M. Blakeley, “Performance Benchmark Test of the Netscape FastTrack Beta 3 Web Server”, Mindcraft Inc., http://www.mindcraft.com/services/web/ns01-fasttrack-nt.html, 1996.

[15] S. Hadjiefthymiades, and D. Martakos, “Improving the Performance of CGI compliant Database Gateways”, Computer Networks and ISDN Systems, Vol. 29, No. 8-13, 1997.

[16] S. Hadjiefthymiades, D. Martakos, and C.Petrou, “State Management in WWW Database Applications”, proceedings of IEEE Compsac ’98, Vienna, Aug. 1998.

[17] S. Hadjiefthymiades, D. Martakos, and I. Varouxis, “Bridging the gap between CGI and server APIs in WWW database gateways”, Technical Report TR99-0003, University of Athens, 1999.

[18] S. Hadjiefthymiades, S. Papayiannis, D. Martakos, and Ch. Metaxaki-Kossionides, “Linking the WWW and Relational Databases through Server APIs: a Distributed Approach”, proceedings of AACE WebNet ‘99, Hawaii, October 1999.

[19] “Performance Benchmark Tests of Unix Web Servers using APIs and CGIs”, Haynes & Company - Shiloh Consulting, http://www.tedhaynes.com/haynes1/bench.html, November 1995.

[20] “Informix-ESQL/C Programmer’s Manual”, Informix Software Inc., 1996.

[21] IS 9075-3, “International Standard for Database Language SQL - Part 3: Call Level Interface”, ISO/IEC 9075-3:1995.

[22] A. Iyengar, “Dynamic Argument Embedding: Preserving State on the World Wide Web”, IEEE Internet Computing, March-April 1997.

[23] “JDBC Guide: Getting Started”, Sun Microsystems Inc., 1997.

[24] N. Yeager, and R. McGrath, “Web Server Technology - The Advanced Guide for World Wide Web Information Provides”, Morgan Kaufmann Publishers, 1996.

[25] D.Kristol, and L.Montuli, “HTTP State Management Mechanism”, RFC 2109, Network Working Group, 1997.

[26] J. Laurel, “dbWeb White Paper”, Aspect Software Engineering Inc., August, 1995.

[27] “Data Management: SQL Call-Level Interface (CLI)”, X/Open CAE Specification, 1994.

[28] R. McGrath, “Performance of Several HTTP Demons on an HP 735 Workstation”, http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html, April, 1995.

[29] “Internet Server API (ISAPI) Extensions”, MSDN Library, MS-Visual Studio ’97, Microsoft Corporation, 1997.

[30] “ODBC 3.0 Programmer’s Reference”, Microsoft Corporation, 1997.

[31] M. Tracy, “Professional Visual C++ ISAPI Programming”, Wrox Press, 1996.

[32] “Writing Web Applications with WAI - Netscape Enterprise Server/FastTrack Server”, Netscape Communications Co., 1997.

[33] “User's Guide, WebDBC Version 1.0 for Windows NT”, Nomad Development Co., 1995.

[34] “CORBA: Architecture and Specification”, Object Management Group, 1997.

[35] “Programmer’s Guide to the Oracle Pro*C/C++ Precompiler”, Oracle Co., February 1996.

[36] R. Orfali and D. Harkey, “Client/Server Programming with JAVA and CORBA”, Wiley, 1998.

[37] P. Ju, and Pencom Web Works, “Databases on the Web - Designing and Programming for Network Access”, M&T Books, 1997.

[38] W.R. Stevens, “UNIX Network Programming”, Prentice Hall, 1990.

[39] M. Leventhal, D. Lewis, and M. Fuchs, "Designing XML Internet Applications", Prentice Hall, 1998.

[40] C. Petrou, S. Hadjiefthymiades, and D. Martakos, "An XML-based, 3-tier scheme for integrating heterogeneous information sources to the WWW," proceedings of the Internet Data Management workshop (IDM '99), DEXA'99, Italy, 1999.

[41] D. Box, G. Kakivaya, A. Layman, S. Thatte, and D. Winer, "SOAP: Simple Object Access Protocol", Internet Draft, <draft-box-http-soap-01.txt>, November 1999.

[42] V. Turau, "The DB2XML user manual (Version 1.1)", Technical Report TR-01-99, FH Wiesbaden, May 1999.

[43] M. Fisher, "The JDBC 2.0 Optional Package", http://java.sun.com/products/jdbc/articles/package2.html, January 2000.