

# A Real-time Web/ORDB Server Implemented using IP-based Object Methods

*As online traffic and the demands for information increase, the Quality of Service (QoS) delivered by transaction-intensive web sites will depend on high-speed fault tolerant Web/database server technologies. The most efficient Web technology-based real-time systems will provide standard web browsers with direct access to data running in main memory as well as fault tolerance for transaction reliability and continuous operation. In addition to direct web browser access and fault tolerance, the Web/database Server must support distributed applications to provide scalability, load balancing, or use as a real-time processing component to any on-line information system.*

## FOREWORD

**A**s online traffic and the demands for information increase, the Quality of Service (QoS) delivered by transaction-intensive web sites will depend on high-speed fault tolerant Web/database server technologies. The most efficient Web technology-based real-time systems will provide standard web browsers with direct access to data running in main memory as well as fault tolerance for transaction reliability and continuous operation.

In addition to direct web browser access and fault tolerance, the Web/database Server must support distributed applications to provide scalability, load balancing, or use as a real-time processing component to any on-line information system.

Polyhedra's built-in classes for TCP/IP and UDP/IP support the use of object methods, based on any IP-based protocol, for developing real-time access to online database applications including HTTP-based web servers, domain name servers, NNTP news servers, remote boot servers, and SMTP mail generation and handling. The database can even include direct support for SNMP.

The Polyhedra Object-relational database contains object methods that communicate with browsers using HTTP, with object methods running in the database process itself and accessible to other objects/information stored in the database. This allows the database, for example, to act directly upon HTTP messages providing direct, real-time responses to web browsers with dynamically created pages from data processed in main memory.

In addition to this powerful capability, Polyhedra's Active SQL Query mechanism provides a fine-grained SQL Push Technology. Active Queries, allow SQL statements, and low-level object queries, to be persistent and send web clients any changes in the original result set. This allows web browsers accessing a Polyhedra web/database server to receive the "deltas", including all individual attribute changes of any affected records. This feature ensures that browsers never

need to poll the server in order to keep up with changes. Using an Active Query and, say a Java-script application on the browser, a browser client can be continuously sent any changes of an object within Polyhedra, for example, the price of a stock or the temperature of a machine.

Even on the Internet itself, a main-memory web/database server can provide a significant performance advantage. An e-business system based on this architecture will deliver significantly better performance to a greater number of concurrent online browser clients than conventional systems with separate passive databases and web servers. Polyhedra's web site at [www.polyhedra.com](http://www.polyhedra.com) provides an example.

CL, Polyhedra's internal object-oriented method scripting language, has been used to implement the HTTP, SMTP and NNTP protocols as object methods. The web server is a Polyhedra database that contains all of the information viewed online and uses no other web server technology. Pages are dynamically created on the fly. For Polyhedra's own Intranet access, the server offers tables of web access history and sales information requests. Information requests are emailed automatically to personnel with acknowledgements emailed back to the requestor. In fact, any Polyhedra database application built with this object-oriented technology can be accessed via a web browser. Even laser printers have IP addresses nowadays!

## TYPICAL WEB SERVER TECHNOLOGY

A web browser is a program that can gather and display information from 'web servers'. To do this, it connects via TCP/IP to the server, issues a request (using the HTTP protocol), and interprets and displays the results. The 'files' delivered by the web server are typically textual, in HTML (or XML) format, or graphical, in which case they will usually be graphics in GIF, JPEG or BMP format. The HTML files contain hypertext markup instructions: not just the raw text to be displayed, but also instructions on how the text is to be

***AD RTS PARIS***

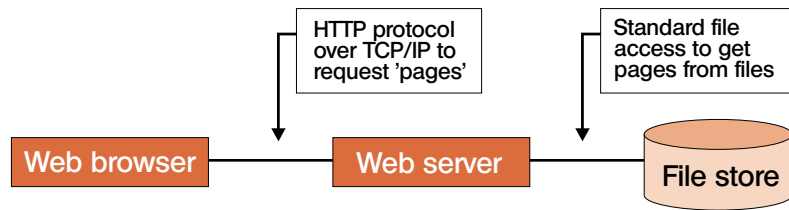


Figure 1.

laid out, the graphic images to display in the document, and the action to be performed when various links are to be followed.

As shown in Figure 1, most web servers get their data - the graphic or textual 'pages' they deliver to the web browsers - from individual files on their local disks.

Designing a typical web site consists of setting up a server to make a selected directory accessible, and then populating that directory and subdirectories with HTML files and graphical images. Where dynamic pages of data are wanted, though, simple and direct file access does not suffice. Active Server Pages (ASP) can be used as a dynamic page server mechanism but it typically requires ODBC for database access. This way to get a web page, where some or all of the contents are generated as the result of a SQL query, is for the web server to run a separate program which reads a template file, connects to the database then queries the database. Finally, it uses the template and the results of the query to dynamically generate the page(s) of information and returns it to the browser. (See Figure 2)

A separate application - often referred to as a 'cgi' program - is started up each time one of the dynamic pages is requested by a browser, which obviously affects performance. The various layers and complexity of code in typical ASP based dynamic web servers affects performance as well as development and maintenance effort.

## POLYHEDRA WEB DATABASE-SERVER TECHNOLOGY

Polyhedra supports object-oriented development with class inheritance, encapsulation, persistence, polymorphism, etc. and an internal scripting language for defining application functionality or "object methods" that can be included in with the data record that defines each object. Polyhedra servers are viewed and accessed as "Relational Object Stores" with SQL access, manipulation and definition provided with SQL that has been extended to support inheritance, methods, and user-defined datatypes among other things. In addition to snapshot data persistence, Polyhedra provides a sophisticated Historian for high-speed data journaling and storage to disk. The data stored via the Historian is accessible via static and active SQL.

To support continuous operation, fault tolerance is provided with an Active and redundant Standby database configuration. A standby database is kept up to date with the changes that occur in the active database, and is ready to go live as soon as it is told to do so - either as a result of a system crash or because a controlled switch-over was requested.

The roles of the databases can be switched at any time and Polyhedra provides several configurations to decide which of the two databases should be Active and which should be Standby. The Standby database is on-line at all times and accessible to clients for read-only queries.

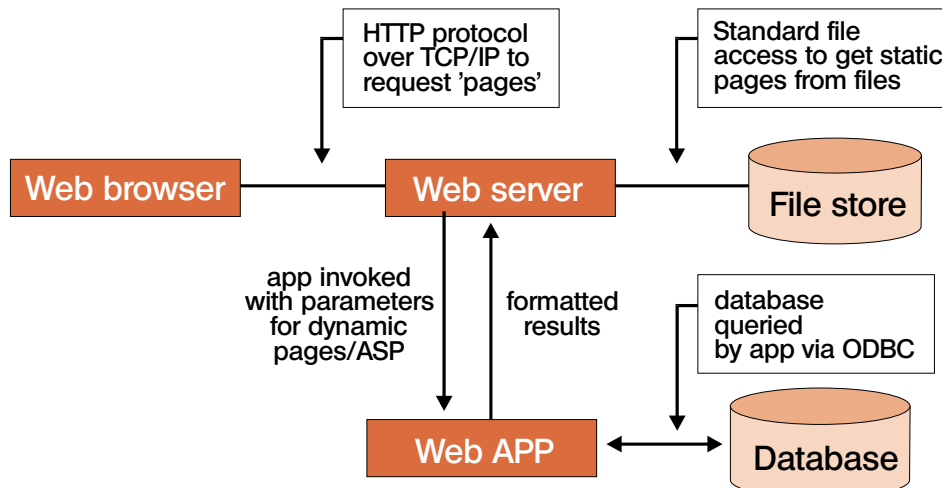


Figure 2.

With the active Object-relational nature of a Polyhedra database, new and more powerful approaches to building web database servers are possible. The simplest is to use the built-in class support for TCP/IP and define an object in the database that allows the whole database to act as a web server. Individual tables (classes) can contain more specific object methods. Any IP-based protocol, for example SNMP or SMTP, can be used as an object method for any individual objects (Figure 3).

To illustrate how this can be done, and to simplify the task of database developers that want to make their database web-aware (visible to online browsers), a sample set of HTTP web server support classes is included in the standard Polyhedra demo pack (in the directory poly/demos/http09).

The files http09.sql and http09.cl between them define two classes: httpServerBase, which handles incoming connection attempts by browser clients, and httpConnectionBase, to handle individual connections. When it sees a web browser is trying to connect to the database process, an httpServerBase object creates an httpConnectionBase object. It is the httpConnectionBase object that reads and analyses the request, and generates and sends the response directly to the browser.

To be more accurate, the httpConnectionBase calls a virtual method called 'respond' to work out what the response should be, and the results of this method are returned to the web browser. The default definition of respond does nothing, however. The database designer derives his or her own class from httpConnectionBase and redefine the respond method, as is shown in another of the standard demos (poly/demos/currency). In the currency example, the respond method is redefined so that:

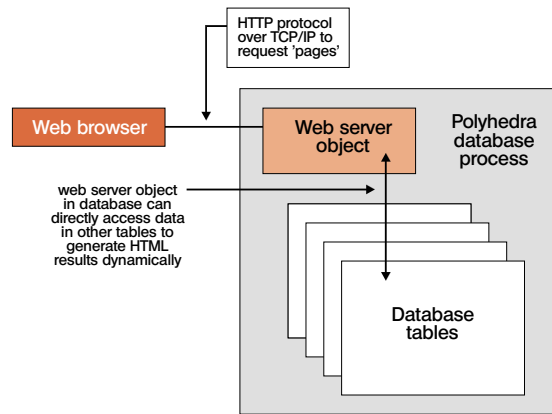


Figure 3.

- the requests "GET /" and "GET /home.htm" iterate over a table in the database, and return an HTML page consisting of a table that lists the name and value fields for each record found in the database table;
- a request of the form "GET /<name>.htm" will locate the corresponding record in the database table of interest, retrieve historic information about earlier values of the record, and return the results in tabular form; and,
- any other request will result in an error message being returned.

The code for performing this is provided in the appendix and consists of about 100 lines of CL. Adapting a new Polyhedra-based application to become web-visible requires only the definition of this function (whose complexity depends on the complexity of the views

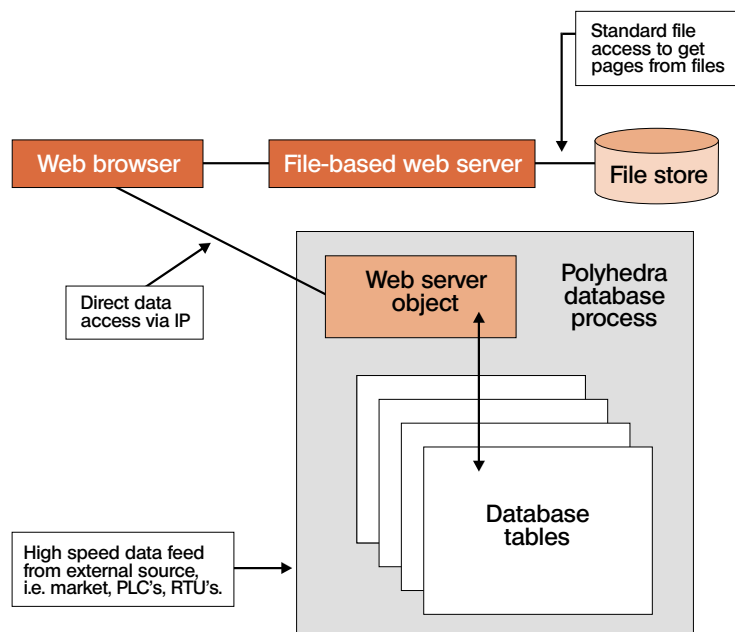


Figure 4.

# INTERNET

required) and of a further 20 lines of CL and SQL (again, see appendix).

How would one use such a system in a live application? There are a number of choices. You could make the Polyhedra database the main web server, by adding in classes for holding the static pages. An alternative is to have a traditional file-based web server, such as Apache, hold the static data and have the Polyhedra database hold dynamic data where intensive query resolution and transactions are performed and access to changing data in real-time is required. Client browsers that start active queries, for example to track the movement of stock prices in real-time, will access the Polyhedra web server directly. As a high-speed data acquisition system, Polyhedra can handle data feeds from external sources such as the Market or remote devices and provide this data in real-time directly and cheaply to online browsers. (Figure 4)

In order to protect the Polyhedra database from potentially-malicious browsers, it is possible to 'hide' the Polyhedra web server behind the other web server, which would act as a filter/cache for the Polyhedra web server; users would not be aware that certain of the pages were being created by a separate web server.

Polyhedra is processor, operating system, and TCP/IP stack independent. It can be used as a web server/database on any operating platform supported by Polyhedra to include VxWorks, LynxOS, OSE Delta, and pSOS, popular UNIX implementations, Linux, and Windows NT.

## APPENDIX: CODE EXAMPLE

To let an application make use of the demo http09 code, it is necessary to pick up the http09 code, to define two classes derived from those defined in the http09 directory, and to create a web server object to handle connection attempts. The SQL could be as follows:

```
include './http09/http09.sql';
create schema
create table httpServerCurrency
( persistent
, derived from httpServerBase
)
create table httpConnectionCurrency
( transient
, derived from httpConnectionBase
, hist_data array of history -- for use by 'respond'
)
-- add an extra class to hold retrieved historical data:
create domain history
( transient
, granularity integer
, starttime datetime
, value real
)
; -- << -- end of schema
insert into httpServerCurrency (id, port) values (1, 80); commit;
```

The CL will consist of two scripts, one for the httpServerCurrency table (telling it to create an httpConnectionCurrency object when a browser tries to connect) and one for httpConnectionCurrency to redefine the Respond handler:

```
script httpServerCurrency
function httpConnectionBase MakeConnection(integer address,
integer id)
--
-- return a pointer to a suitable object derived from
-- the class httpConnectionBase
--
local reference httpConnectionBase obj
create httpConnectionCurrency \
( id = id \
, owner = me \
, address = address \
) into obj
return obj
end MakeConnection
end script
script httpConnectionCurrency
function binary Respond
--
-- analyse request, return response as a binary value.
--
local integer n
local binary b
local string str
local string url = word 2 of request
local string title = ""
local reference tables t
local reference object obj
local reference currency c
local reference history h
if url = "" or lowercase (word 1 of request) o "get" then
set str to "<H1>Unknown request</H1>" & \
"<P>Sorry, cannot understand request" & \
request & ""
else if url = "/" or url = "/home.htm" then
set title to "<HEAD>TITLE: Currencies</TITLE></HEAD>"
set str to \
```

Article by Polyhedra.



**Dedicated Systems**  
**Encyclopaedia**

**RTOS BUYER GUIDE ONLINE**  
**&**  
**CALENDAR OF EVENTS**

at the Dedicated Systems Encyclopaedia web site:  
<http://www.dedicated-systems.com>