Design Patterns for CGI Web Applications with Visual Basic

Mike Lopez, John Peppiat
Manukau Institute of Technology
Auckland, New Zealand
Mike.Lopez@manukau.ac.nz

ABSTRACT

Many commercial organisations want to use the Internet to extend the reach and richness of their interactions with business partners. The choice of the right technology to use is complex and there is little practical data to support decisions. Current technologies require compromises to be made between the needs of performance, robustness and the skills of application developers.

This paper demonstrates how to develop robust, medium-scale CGI database applications in VB 6 and describes the results of simulations. The authors show how the use of their innovative class library and design architecture makes writing VB-CGI applications a straightforward matter - something previously thought impossible. Previous approaches (Denny 1997) have not addressed the need for encapsulation or for the support of n-tier architectures.

The authors contend that the technology described has adequate performance to support typical Web trading applications using inexpensive hardware and software, is easier to work with than PERL or C, can be readily understood by the large community of Visual Basic developers and facilitates the use of robust design patterns.

1. INTRODUCTION

Common Gateway Interface (CGI) applications are designed to process input sent to an Internet server by an HTML form. The full CGI specification can be found at the National Center for SuperComputer Applications (NCSA, CGI V1.1). They are usually developed as compiled executables created in a language like C or PERL. The key element of this architecture is that the CGI application and the Web server run as separate processes, each maintaining its own resources and interfaces. To process a message the Web server creates the CGI process, passes information to it via environment variables or by a standard input file stream, and receives a response as an output file stream. Once the message is handled the CGI process is no longer required and usually terminates.

For the Visual Basic programmer, learning and working in languages like PERL and C is not an attractive option for a number of reasons. Firstly, there is the obvious learning curve associated with a new language and programming environment. Secondly, there is relative complexity of these environments when undertaking common tasks like interfacing to database products or using COM objects. Finally, the programmer will inevitably miss the rich support for coding and testing provided by the VB Integrated Development Environment (IDE). Taken together, such shortcomings lead the programmer to experience frustration and a lack of productivity - especially if the end product performs poorly when deployed.

Given the proven stability of CGI, and the dominance of VB in the programming world, the obvious question is 'can VB be used to write a CGI application?' The answer is a most definite yes. In fact, it is quite straightforward, and once mastered the full capabilities of VB can be exploited to build fully featured, scaleable CGI applications much more quickly than with PERL or C. In many instances, this approach is easier for a VB programmer to master than is the transition to using Active Server Pages (Microsoft's script based technology, ASP).

In the following sections we will describe how VB can be used to develop a CGI application, present a design pattern that addresses encapsulation and scaling issues, and report on the results of some benchmarking trials using this technology.

2. THE MECHANICS OF CGI WITH VB

CGI applications operate by receiving data via operating system environment variables and the default console input channel (known as stdin to C programmers), and produce output on the default console output channel (stdout). VB has no standard project type for a console application, however if the caller (the web server) has set up an associated console then standard windows API calls can be used to access them.

Example code to make such connections is:

```
hConin = GetStdHandle(STD_INPUT_
HANDLE)
hConout = GetStdHandle(STD_OUTPUT_
HANDLE)
```

Example code to read and write is

```
hr = WriteFile(hConout, What,
Len(What), Count, 0)
result = ReadFile(hConin, txtIn, ct,
ct, 0)
```

Two methods are used to send information from the web browser, GET and POST. Conveniently, the CGI standard uses an environment variable to inform the CGI application of the method used to submit data. This makes it possible for the user to write code that is independent of this distinction.

2.1 Get Method

One method of sending information from a web browser program to a CGI program on a Web server is to construct a URL request formatted in a prescribed manner. Example:

```
http://xyz@domain.com/program.
exe?name=fred
```

The name value pairs (e.g. name=fred) are sent as a single environment variable called QUERY_STRING that can be accessed using the VB environ$ function and parsed by code.

Example

```
s = Trim$(Environ$("QUERY_STRING"))
```

2.2 Post Method

The alternative method used by HTML forms is POST. In this case the name value pairs are placed in a simple document similar to a .INI file and transferred to the CGI program as they standard input file. In this case the CGI must read and parse its standard input.

2.3 Encapsulation in Classes

A set of VB classes was designed to support VB CGI programming. They expose a simple interface to the developer, and encapsulate the low-level logic for processing input and packaging output. To illustrate the ease of the class support the following simple CGI provides a hello world response

```
Dim CGI as CGIClass
```

```
Set CGI = New CGIClass
CGI.Text = "Hello World"
```

3. SCALEABLE DESIGN PATTERNS

A simplistic approach to producing a CGI application is to construct it as a fully self-contained application. In a typical commercial application, this would involve processing the input from the form, manipulating a database and producing HTML text as a response. This monolithic approach undoubtedly works but has some drawbacks both from performance and architectural viewpoints.

Response time is a function of the overall complexity of the message processing, but performance bottlenecks for any database CGI application include the time taken for the Web server to create the CGI process and the connection time to external services like the database. The physical size of a compiled VB project is frequently much smaller than its C equivalent. The reason for this is that the VB run-time system resides in DLLs. These DLLs are rapidly cached and the load time of a VB executable is very fast. Having to connect to a database service is however a major overhead but there are ways around this!

A monolithic CGI program runs contrary to common programming practice that emphasizes architecture with clear boundaries between presentation code and business logic code. Seen from this perspective the construction of HTML is clearly the presentation layer, and the database processing is the business logic.

VB supports a powerful technology for constructing and testing a better design pattern, ActiveX, one of the Component Object Model (COM) technologies from Microsoft. The full COM specification can be found the official site for Microsoft's COM specification (Microsoft COM specification, 6/1/2000).

3.1 ActiveX

For the purposes of building an efficient CGI application what is required is to have a CGI process with a small footprint that can quickly connect to some pre-loaded code. VB provides unparalleled simplicity in its ability to develop and manipulate in-process (ActiveX DLL) and out-of-process (ActiveX EXE) COM objects. The useful feature of Microsoft's COM technology is the ability for one process to use an object that resides in another process. The marriage of VB CGI and ActiveX provides an ideal solution to both the performance and architectural problems. ActiveX EXE components are particularly useful

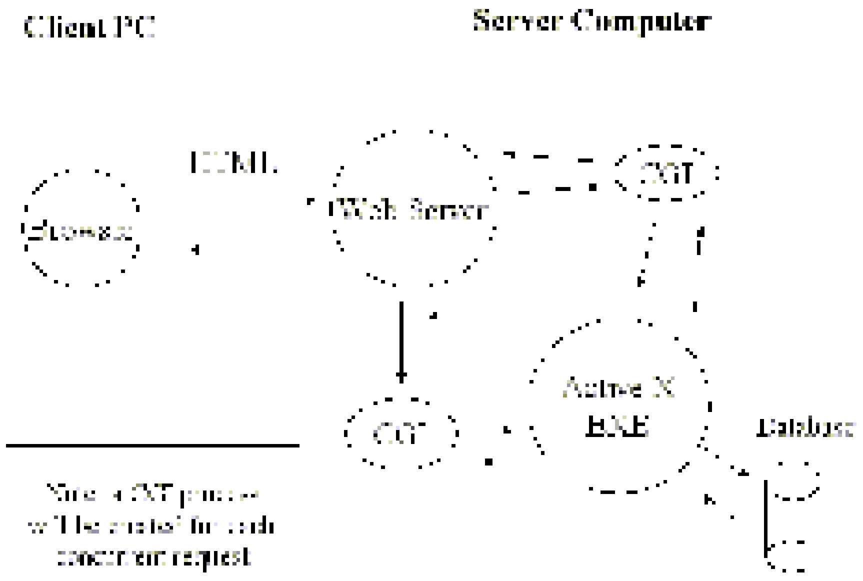

Figure 1:

Depicts the relationship of the various components of this architecture.

where database manipulation is required as they can be pre-loaded and more importantly they can maintain database connections to support incoming CGI requests.

4. EVALUATION

As a means of illustrating the concept, and for the purposes of examining the actual performance of this type of CGI application, the following stylized system was constructed.

An Access 2000 database consisting of a single table called Hits was created. The task of the CGI application was to generate a key for a record in Hits and then update (or create) the record with that key. Once completed, the CGI returns a short acknowledgement message.

The code to maintain a permanent connection to the database and perform the database logic was placed in an ActiveX EXE. The CGI program simply connects to the ActiveX object, invokes the update code and returns a response. For this evaluation, the ActiveX was single threaded which is probably sensible when using a simple product like Access. In a more meaningful application, ActiveX objects could be created to say query products, validate customer credit, place orders and so on.

4.1 Measuring the Performance

In order to examine the performance of the CGI application, a simple VB Internet client was created using Microsoft's Internet Transfer Control. It repeatedly sends a request to a Web server and measures the average response time. Parameters in the client process and the responding CGI were used to isolate the elements contributing to performance. The Internet Client could be deployed on one or more client PCs as required.

4.2 Performance Results

Using these techniques, benchmark tests were conducted using two 730 MHz PC's, one as the client and the other as server. The software involved was Windows 2000, and the personal IIS web server. The results are shown below and summarized in Figure 2

Element	Millisecs
Turnaround time for basic CGI (CGI)	29
Out of Process COM object (COM)	23
Set up ADO connection (ADO)	45
Open recordset (OPEN)	6
Find record in recordset (FIND)	20
Update or add record (UPDATE)	4
Add record via SQL insert (INSERT)	8

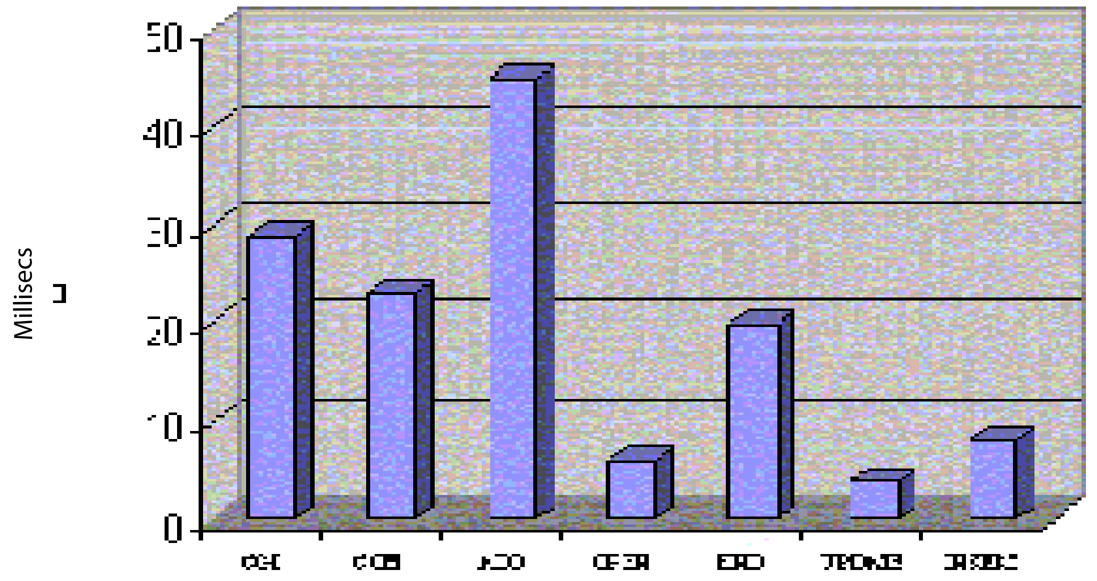

Figure 2

4.3 Robustness

The weak point with n-tier design patterns like that proposed is the Active-X exe which is a single point of failure. Errors encountered in this component have the potential to block the CGIs. The most likely cause will be database interaction, especially if the database is being shared with other applications. The authors recommend that short timeouts are set for CGI responses and database operations, and that all interactions with the database are carried out in SQL rather than through recordsets. To minimize the chance of conflicting interactions, the database should be dedicated to interactive web applications. With attention to these points the authors contend that desktop databases can be made sufficiently robust for the described use.

5. CONCLUSION

In the trial, the most significant contribution to response time was the time to connect to the database using ADO. The ability to use an ActiveX EXE to maintain a cached connection significantly reduces this, and the overhead of making a COM connection to the EXE is significantly less. The proposed N-Tier architecture will out perform a monolithic approach. The use of VB classes to encapsulate the mechanics of the CGI interface enables even a relatively inexperienced VB programmer to write efficient Web applications quickly. Using relatively low specification hardware, and commodity software (Access) transactions of three to four transactions per second should be achievable and reliable.

REFERENCES

Denny R, 1997 WebSite Professional, VB 4 Server Side Programming, http://solo.dc3.com/vb4cgi.html

Microsoft COM specification, 6/1/2000 http://www.microsoft.com/com/.

NCSA, CGI v1.1 specification http://hoohoo.ncsa.uiuc.edu/cgi/interface.html.