An Internet Banking Framework with Perl

Carlos de la Guardia

Javier Rodríguez

Aldea Internet

Abstract

Aldea Internet Banking Framework is a Perl-based Web application that allows financial institutions to provide to its customers a wide spectrum of online banking services, including home banking, enterprise banking and virtual points of presence. Architecturally, the framework integrates seamlessly to the bank's legacy systems through a Communications Module, and allows the integration of improved customer services by encapsulating most of the business logic in a separate Operations Module. As an application, its ultimate goal is to complement -and eventually replace- the expensive online banking in-house developments based on closed or proprietary technologies.

This paper describes the architecture and inner workings of Aldea Internet Banking Framework and the finer details of its implementation for one of the largest banks in México: Banco del Atlántico.

The project

Our customer, Banco del Atlántico, had an application, developed in-house, which allowed its corporate clients to process transactions and consult a number of reports generated on the fly, like balances. This DOS-based application connected via modem to a terminal server located at the bank and from there talked to a mainframe application which actually carried out the desired operations.

The problems with this approach were many: low security, modem and phone line needed for every client, high installation costs and times (because every new version had to be installed all over again), incompatibilities with different versions of the operating system, maintenance nightmares with the code and unhappy clients.

The bank decided to use Internet technologies to solve these problems and at the same time begin the transition from their old, inefficient systems to more modern developments. They also came up with a bold idea for minimizing the cost and resources required for opening new bank offices.

The political situation in the bank at that time made necessary the division of the project in three phases. The first one was the creation of a front end for their current systems that substituted the current client program for a web browser and some server side applications and integrated security measures. The second one included the modifications of this front end necessary for dealing with point of sale equipment, so that it could be used as a mini-bank inside video rental stores equipped with just a computer, modem, bar code scanner and printer. The third phase was the gradual substitution of the old mainframe applications for newer systems developed from scratch to fit the new paradigm.

This paper deals with the first two phases of the project, in which Perl played a major role.

The challenges

From the beginning, the project presented a lot of difficulties. The time frame given to us was rather small (only eight weeks from conception to implementation) and the application was meant to be in production by the fall of 1997. Many bank officials considered the project a wild goose chase. The mainframe applications with which had to interface were not very well documented. And finally, the people responsible for them were seldom if ever at hand.

In México, most banks have their own in-house development teams, and there are no standards for applications programming or file formats. Also, since the banks in México were nationalized in the 80's and re-privatized some years later, there have been a lot of changes in systems management and policies, together with a high personnel rotation.

All this has created a situation in the banks where a consultant is faced with extremely old systems which run a variety of applications coded in different languages and very poorly documented (no source code available in many cases). Banco del Atlántico was a good example of this situation.

Another problem that we faced was this: the existing application used a session-oriented message exchange protocol, so that clients did all their work in a single connection. Of course, the web protocol is a stateless one, so we needed a way to maintain state between web connections that at the same time tricked the mainframe application into thinking that everything was happening in a single session.

In short, we were presented with the following challenge: create a middleware capable of communicating with a session-oriented COBOL application running in a UNISYS mainframe, which then could interface with a stateless web server to present the old information in a new, attractive format in a web browser.

The approach

We decided early on to use a modular approach for the construction of this project. The idea was to encapsulate tasks like communication with legacy systems and business logic into separate modules. This would hide the implementation details from the main application and each other. This way, if changes were made to some part of the system in the future (say, the COBOL application gets updated or a new machine is added), only the pertinent module would need to be altered.

To solve the 'stateless protocol' problem, we decided to redirect all web connections to persistent processes that kept an open session on the mainframe for each client.

System requirements

- *Hardware*. The bank decided that the main application would run in an HP server running a secure environment for web servers called Virtual Vault, which would act as a web application gateway.
- *Programming Language*. When deciding upon the programming language to use for our modular approach, we initially considered "raw" C & C++, Java and Perl. However, we quickly saw that C or C++ were not really an option, since the time frame was way too narrow. Java was not considered mature enough because servlet technology was still in early beta. Since both authors are long-time Perl hackers, after an objective language comparison we weren't surprised to learn that Perl was chosen as the programming language for this project. We knew from first-hand experience that Perl allowed rapid development cycles and also how easy it is to construct and integrate modules with it. We also decided at this point to use the wealth of modules already available for Perl programmers to shorten development time even further.
- *RDBM*. Since the bank was in the process of adopting Informix as RDBM of choice, we had to use this database for our project. The DBI modules came in handy here, as they allowed not only a seamless connection to the database but database engine independence as well.

Tools

- *Perl*. Not only the application itself, but also all our testing and monitoring tools for use during the development, testing and deployment phases were written in Perl.
- *FastCGI*. We needed a proven protocol to provide connectivity between the web server and the application gateway. It also solves beautifully the session-oriented vs stateless issue.

Extensibility

One of the main goals of the project was to have the ability to implement and integrate new business functions easily. The key to this was the modular approach. Our design was made in a way that allowed new procedures to be effortlessly inserted into their place as if they always had been part of the original design.

Scalability

Since the plans of the bank were to eventually offer their Internet banking services not just to corporate clients but to would be home bankers everywhere in the country, we had to plan for huge numbers of users from the very beginning. We decided to use a process manager that would route petitions to the first available server process, so that new servers could be added easily as the number of customers required it. Initially only one server would be used.

Security

We're talking about a bank with a direct connection to the Net here. This mere tough sends shivers down the spine of most people on the financial market. But we counted on our experience in Internet programming and on a certain HP product named "Virtual Vault".

Export restrictions were much of a concern, too. Even though the US Department of Commerce allows the export of domestic-level encrypting servers out of the US for use by (some) foreign financial institutions, the browsers available at the time outside the US were unable to interact with a domestic server product using full-strength cryptography. This restriction proved to be unacceptable to the customer.

In order to solve this issue, a third-party product called SeguriProxy was integrated to provide full-strength encryption of the channel trough a proprietary protocol.

This security solution introduced another problem for us, since the Virtual Vault OS is really picky about what things can run in its protected area. We solved this by using a small C program residing in the Virtual Vault's sandbox as a bridge between the web server and the real web application.

A network diagram showing the role of the Virtual Vault is shown if Figure 1.

Architectural overview

Figure 2 shows the different components involved in the application.

- *Web browser*. The application was designed to be browser-independent, but it took advantage of a decent subset of HTML and a couple of javascript code snippets in order to improve the browsing experience. In order to take advantage of full-strength encryption, the option of proxy connection was mandatory.
- *Channel encryption*. We achieved confidentiality and authentication through full-strength encryption using a third-party product split in three parts: a user proxy that ran in the user's box and talked to a central proxy running in the external compartment of the Virtual Vault. Both parts needed to authenticate to each other using a valid certificate in order to establish a secured connection, and the later that would act as a second proxy and connect locally to a restricted, unsecured HTTP server running on a nonstandard port. Finally, a central Certificate Authority would extend both user and server certificates in PKCS7 format,
- *Authorization server*. Its main function is user certificate authentication against a valid-user database. The idea was that it would evolve into a full user clearance system.
- *Process manager*. A FastCGI application itself, this little program reads the user-certificate HTTP header in order to assert its validity. Next it determines if there is a FastCGI process running for that user, and creates one if necessary. After that, it connects the incoming HTTP connection to the corresponding FastCGI session.
- *Application gateway*. The FastCGI application would encapsulate the user interface logic, the business logic, the protocol-session state machine and TCP/IP host connectivity in a single, isolated process. This process would receive all kinds of HTTP headers and input – and emit the corresponding output- through process manager as if it were directly connected to the corresponding customer's browser.

Why Perl?

First of all, we believe in Perl. We know what it can do and have been promoting its use for important projects since we started working in this field. Also, we like Perl and would rather work with it that with other languages whenever possible.

But most important, the particular conditions of this problem made Perl the perfect solution. Which other language would allow to turn in a successful application in a short time frame and with less than optimal working conditions? Which other language offers such an open repository of proven tools capable of greatly reducing testing and development cycles?

Advantages of using Perl

- *Short development cycle*. Code modifications and the implementation of new functionality were easily incorporated into development code and got to production in almost no time without the need to recompile each time. Besides that, the development facilities of Perl are hard to rival: all the way from warnings and tainting to the integrated Perl debugger, the language contributes to make the programming experience not just easy, but also fun.
- *Maintainability*. The use of Perl modules along with the Revision Control System allowed a stable and maintainable code base. The self-documenting capabilities of the language and a coding style that resembled the english language also allowed easily readable code.
- *Extensive use of language features*. The use of regular expressions in the parsing of server messages proved to be quite a time saver. The DBI and DBD::Informix modules permitted an easy database integration, as the networking modules were of great help.
- *Code reuse through the usage of existing modules*. We were able to use existing modules (Most notably CGI.pm and Telnet.pm) and implement a top-down modular architecture using Perl modules.

Drawbacks

- Obscurity of language. In corporate IT circles, anyway. Need to offer Perl training as part of the bundle. Lack of commercial support is a common worry. Perl advocacy docs came in handy here.
- Performance. Worries about expected performance under heavy load. Considerable footprint (~1.1MB per user) . Multiprocessing instead of multithreading. (Hey, we look forward to Perl multithreading!)

Implementing persistency: the application core

Since the core of the developed applications belong either to the bank or to Aldea Internet, which is our company, we are not at liberty to give away all the code from our work. However, Aldea has allowed us to freely distribute the code from our process manager, which in many ways was the heart of our solution. This module is available electronically at http://www.aldea.com.mx/papers/banking/ .

Anyway, the majority of the modules developed for this project are too specific for our client and not of real use for the general Perl community.

We feel that the process manager could be a really useful tool for those who want to interface with session based legacy systems using stateless protocols like HTTP. Due to the time limits set for this project the tool is far from completed and could maybe be used more as a model than as a finished application.

Some ideas we came up with for this program during development are:

- Interface for serial or terminal based systems.
- Interface for complex legacy database driven applications.
- Interface for a MUD system.

We have found in our experience that the first two problems mentioned are fairly common in the financial sector when dealing with legacy applications and their integration to new technologies like the web.

Conclusions

This project was very successful for us and proved to many unbelievers that Perl can really offer solutions to complex problems in the real world of financial institutions. Not only did we save time and money using Perl, but also developed a strong and scalable solution with minimal recoding because of Perl's strength joined with its unmatched properties as a 'glue' language.

***Carlos de la Guardia** is Director of Systems Development at Aldea Internet. He has five years experience in developing Internet applications. He can be reached at Aldea Internet, Daniel Delgadillo #5, Tlalnepantla, Estado de México, México.* CP 54050. cguardia@aldea.com.mx or http://www.aldea.com.mx/~cguardia/.

***Javier Rodríguez** is Research and Development Manager at Aldea Internet. He received a BS in electrical engineering from ITESM Campus Estado de México and has been working on Internet Application Development for the last six years. He can be reached at Aldea Internet, at* jrodrig@aldea.com.mx or http://www.aldea.com.mx/~jrodrig/ .

An electronic version of this document is available at http://www.aldea.com.mx/papers/banking/ .