White Paper:

The Curl™ Platform: A Deployment Architecture for Rich Client Applications

Table of Contents

Introduction 1
Architecture Overview 2
  Motivation 2
  A Deployment Architecture Optimized for Web Performance 2
Deployment Architecture 3
  Stable, High Performance Runtime Environment 3
  Secure Mobile Code 5
  Web Standards for Networking and Data Connectivity 5
  Disconnected Operation 6
  Dynamic Generation 6
Conclusion 7

Introduction

While the Web has enabled an unprecedented amount of connectivity and centralized data distribution, it has also resulted in a return to the days of a “dumb terminal” interface with limited interactivity and vastly inefficient use of bandwidth. This is because the Web (and the HTML standard which enabled its proliferation) was originally designed to transmit large amounts of text-based data with minimal presentation logic; a few hyperlinks and images, perhaps, but not the extensive layout capabilities and interactive elements demanded by Web applications today.

The solution to this problem is creating a way of maintaining presentation and associated business logic on the client, which requires that the client be more than a “dumb terminal”. It must be a live, executable application that is capable of interacting with the end user, retrieving additional data for display, and storing data for manipulation, filtering, or re-presentation. The challenge is to provide this rich client interface without reverting to the days of installed client-server applications, with all of the associated configuration issues and high costs of ownership. This requires the use of “mobile code”, executable applications that can be delivered over a network to the client device.

The Curl™ platform was designed from the ground up for rich client Web applications. This has resulted in a unique language and deployment architecture optimized for Web delivery, addressing both the limitations of present Web technologies and the deployment issues of other rich client approaches.

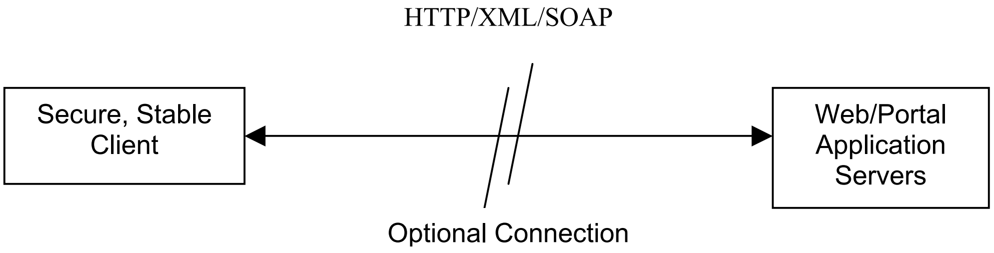
Architecture Overview

Motivation

Back when the Curl project was started in 1995, many of the computer scientists at MIT had already been using HTML (and later, JavaScript) for 6 years. While these standards were fine for Web documents with limited interactivity, the founders of the project realized that for true Web applications they needed a more fully-featured, robust technology. Curl technology was a result of this project, providing a basis for the deployment architecture outlined below.

A Deployment Architecture Optimized for Web Performance

Web delivered applications have traditionally relied upon a thin client deployment architecture which is vastly inefficient from a bandwidth and server processing perspective, resulting in poor usability and negatively affecting user productivity. The solution to this problem is utilizing mobile code, and leveraging the power of the client device as an equal partner for delivering rich client applications. However there have been problems with mobile code approaches of the past; security problems and performance issues being the most well known. The Curl™ platform offers a deployment architecture which solves these mobile code implementation issues. In addition, the architecture is very flexible, supporting simple integration with existing Web infrastructure, disconnected operation, dynamic generation of code, and all of the centralized deployment benefits offered by the Web.



Deployment Architecture

A mobile code architecture has two main components: the code itself, and a client-side execution environment for that code. The Surge™ runtime environment provides a stable, high-performance platform for executing Curl code. It was designed to execute Curl code identically, regardless of browser or platform. It also implements a unique configuration and versioning system that allows multiple versions of the Curl API to run concurrently on the same machine. Additionally, the runtime implements a security model designed to address the unique challenges of mobile code.

There are also several design considerations outside of the runtime which allow for very flexible deployment options. The first is that networking and data connectivity is based entirely on Web standards. The second is that Curl code can be dynamically generated on the server and evaluated on the client. In addition, Curl code can execute even when disconnected from the network, allowing mobile operation previously unavailable in Web applications.

Stable, High Performance Runtime Environment

The Surge runtime environment provides an execution platform that is optimized for Web performance. On the Web, the primary bottleneck is getting the code to the client; a combination of the server's processing time and downloading the code. The Curl platform offers advantages over traditional thin client and rich client Web technologies in both of these areas.

In contrast to traditional Web applications, Curl processing is completed entirely on the client. This alleviates the bottleneck of server performance, allowing for more scalable and robust applications. As for the bandwidth constraint, Curl lends itself to highly abstract and compact code by implementing just-in-time (JIT) parsing and compilation directly from source code. Additionally, whereas traditional Web applications use a thin client approach, Curl code is a real, executable application. This allows the application to run continuously without further downloads other than for data or component retrieval. This vastly reduces download time, increasing user productivity.

While JIT compilers offer many benefits, traditionally they have been negatively perceived in the area of execution speed. The Surge JIT compiler is optimized to alleviate these concerns. It makes extensive use of lazy compilation, executing code fragments only when they are needed by the application. These results in performance for the end user which is nearly indistinguishable from traditional statically compiled applications.

Additionally, the Surge runtime environment is browser and platform agnostic. Most of the runtime libraries are written in Curl, utilizing the core language libraries for all logic and presentation. This allows porting of the runtime across platforms as long at the core functions of the language are supported.

Additionally, the plug-in portion of the runtime allows Curl applications to execute identically regardless of browser, and support both embedded and full page operation.

Configuration and Versioning

Client applications traditionally suffer from deployment problems when installation upgrades replace shared libraries on the client machine. Any programs relying on those libraries often behave unexpectedly or stop working if the new library is not exactly backwards compatible with older versions. Unfortunately, library backward compatibility is a very difficult thing to get right, resulting in a deployment issue known as “.dll hell”.

The Surge runtime environment was designed with a configuration and versioning system to solve this problem. There are two parts of this system: versioning of the runtime libraries installed on the client, and allowing Curl developers to version their applications.

The runtime packages installed on the client machine are versioned to correspond to the version of the API they support. When a new version of the Curl API is released, a totally new set of runtime packages are created. If an application wants to take advantage of this new API, these packages are downloaded and installed alongside the original installation. This works for both newer and older versions of the API, allowing many applications to run concurrently on the same machine, without relying on library compatibility to ensure they work correctly. The bits that the application was written to are the bits that will execute it.

The other piece of configuration and versioning is the ability to specify which version of the Curl API an application or package was written for. This is done in the first line of an applet, know as the herald:

```
        {curl 1.7 applet}
```

After testing to ensure compatibility with multiple API versions, an application author can add compatible numbers to the herald:

```
        {curl 1.7, 2.0 applet}
```

This offers a lot of flexibility for applet authors working in varied deployment environments.

Secure Mobile Code

The Surge security model was designed to eliminate the roadblocks which have hindered the development of secure mobile code applications. This has been accomplished by removing complexity and end user security interaction, learning from the mistakes of the past, and building in alternative mechanisms to expand the power of unprivileged applets. In doing so, the Curl solution allows robust functionality in an unprivileged environment.

A long-held but rarely enforced belief in the security industry: a simple system is inherently more secure than a complex one. There are many caveats, however the fundamental principle is sound: by introducing complexity, one increases the number of possible attack modes, creates a more difficult administration task, and opens up the distinct possibility of human error. The Surge security model was designed with this belief in mind. It was designed to empower an applet to do things that were previously thought to be “outside the sandbox”, but do them in a secure manner, minimizing the use of trust. It was designed to allow system administrators the power to explicitly grant applets access to network resources, rather than forcing them to remove access implicitly granted. It was designed to require a minimum of end user interaction. In a word, the Curl model was designed to be simple.

To be clear, the word “simple” when talking about security does not imply technically deficient. In fact, it is far more difficult to create a simple interface to intricate, underlying security systems than it is to create a convoluted, complex interface. The Curl security model has many features that display a conscious decision to extend the capabilities of the technology while creating a simple, easily-administered security interface.

The traditional sandbox security model placed severe restrictions on an applet’s usefulness. As a result, most applet authors resorted to signing applets, which allowed end users to give the applet trust and break outside the sandbox. While this allowed applets to do interesting and powerful things for end users, it was a troubling problem for administrators. By allowing end users, who rarely make informed security decisions, the ability to grant trust to applets, administrators were effectively putting the security of their network and the PC at risk. Left with few alternatives, many administrators restricted or eliminated the use of mobile code on their network.

In general, the Curl security model dictates that unprivileged applets must be more powerful than traditional sandbox model allows. In doing so, it reduces the need for privileged applets, which are potentially dangerous and have prevented adoption of mobile code technologies in the past. While it is still possible to give an applet privilege, Curl does not provide any mechanism (ie. signing) by which the author can request privilege from the end user. A system administrator must explicitly grant privilege to the applet, and it is highly discouraged. By building a bigger sandbox, Curl allows developers and IT professionals to accomplish powerful tasks while maintaining the integrity of their network and systems.

Web Standards for Networking and Data Connectivity

As a technology for distributed Web applications, a key piece of Curl’s architecture is that Web standards are used by default for all networking and data connectivity. By utilizing HTTP/S for standard data transport, XML for data translation, and SOAP for remote procedure calls (RPC), a Curl application can plug seamlessly into existing Web infrastructure.

Curl's networking API is based underlying support for TCP/IP, allowing privileged applications to create socket level connections for customized networking. The default method of unprivileged transport, however, is HTTP/S. This allows Curl applications to interact with any standard Web, application, or portal server, lowering the complexity of deployment.

Built on top of our standard networking, Curl's XML API allows application developers to interpret data feeds encoded in the popular data exchange format. Of course, one could also write a parser in Curl for any other data format, but this API enables better interoperability by enabling application authors to single source their data repositories, sharing information across the network.

Finally, integrated support the SOAP RPC standard allows both the networking and XML portions of the system to be utilized in more complex distributed environments. Any backend system with a SOAP interface can now be accessed directly by the client (if the system administrator has explicitly granted Curl access). These more distributed architectures can reap significant scalability and cost benefits.

Disconnected Operation

Since Curl applications are JIT-compiled by the runtime, once code has been transmitted to the client it can continue to execute in either network connected or disconnected environment. This is accomplished by saving the application to the client device, and designing the application to maintain a data store on the client. This allows application authors much more flexibility when determining end user usage patterns, allowing them to connect and disconnect from the network without loss in application functionality or user productivity.

Dynamic Generation

JIT compilation also allows for dynamically generating Curl code on the server. This has two benefits. First, it provides an easy migration path for integrating Curl into existing backend infrastructure designed to dynamically create HTML. Most applications, however, can benefit from converting their architecture to achieve the same dynamic effects directly on the client, downloading data and components on demand and increasing responsiveness. Second, back end data sets can be generated as Curl code rather than XML or some other data format. This allows the application to skip the data-parsing step and dynamically compiled the data directly into Curl objects.

Conclusion

There are many reasons that the Curl™ platform provides a compelling architecture for next generation Web applications, however they all stem from the same source: Curl was designed from the ground up for Web applications. This is shown throughout both the innovative language design and the robust deployment architecture. By choosing Curl for a rich client solution, one is able to address development and deployment issues which have plagued mobile code implementations of the past while providing a more rich, productive application for end users.