



AberdeenGroup

Flash Remoting MX: A
Responsive Client-
Server Architecture for
the Web

An Executive White Paper

December 2002

Aberdeen Group, Inc.
260 Franklin Street, Suite 1700
Boston, Massachusetts 02110-3112 USA
Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com

Flash Remoting MX: A Responsive Client-Server Architecture for the Web

Executive Summary

The responsive organization — that is, one capable of anticipating and responding to its customers and to its inventory, as well as to supply dynamics — depends greatly on the ability to access, interpret, and execute against reliable information. This was true before the Industrial Revolution, and it remains true in the aftermath of the Web revolution. Positivists will boast that network computing makes information far more accessible, timely, and reliable than ever, at lower cost. But if infrastructure has steadily improved, the ability of end-users to manipulate it and respond to the resulting fire hose of data has progressed more slowly.

In fact, client-server usability has actually regressed over the last decade, with the displacement of highly functional graphical user interfaces (GUIs) by commodity Web browsers. In its intended role as a universal document reader, the browser succeeds spectacularly: it is standardized across client platforms, relatively lightweight, minimally functional, and therefore easy to administer and use. But as a universal application interface, Web browsers fall short on every criterion but one: low cost. Even in its current scriptable incarnation, the Web's HTML (Hypertext Markup Language) client lacks richness and spontaneity, crippling online responsiveness at its human nexus.

Yet at this point, replacing the browser wholesale with a more dynamic, but unfamiliar portal is impractical. Both at home and at work, browsers have become the preferred client for a powerful array of standardized services, including document and image display, help files, e-mail, and multimedia file transfer. How then can this de facto standard be improved, given the Web's vast installed base on both sides of the firewall? Evolving the browser means extending it in a lightweight, readily programmable way that does not disrupt entrenched economies of scale and skills.

Several technologies have emerged to meet this need, including Dynamic HTML, ECMA-Script (the international standard for JavaScript), and the Document Object Model (DOM). Although each has made the browser a more versatile, programmable client, none of these incremental improvements has resulted in a truly responsive, rich user experience because no native browser technology can alter the fundamentally document-centric, request-response nature of Web communications. At their essence, Web front-ends remain documents, rendered upon each user interaction from a hodgepodge of server-side data and page markup. To regain the spontaneity of client-server software, while untangling presentation logic from event handling and data, a different metaphor altogether is needed.

This Aberdeen *Executive White Paper* examines the requirements for implementing a responsive client-server model on the Internet — without replacing the installed browser base or burdening developers with complex new infrastructure.

The paper identifies Macromedia Flash MX as a rich-client technology that introduces a powerful event-centric metaphor to meet these requirements. Pragmatically, the client-side Flash Player is nearly as widely installed as Web browsers themselves. The second half of the paper describes Flash Remoting MX, a new application server gateway with clear productivity, performance, team workflow, and lifecycle advantages for developers engaged in the design and construction of Internet client-server applications.

Uncrippling the Client

In some aspects, the Web browser has come a long way from its original implementation as an HTML page-rendering engine. Compared with early browsers, which had trouble displaying simple tables and responding to events other than mouse clicks, modern browsers feature more precise page formatting controlled

The basic nature of Web pages and their interactive capabilities remain limited by the browser's request-response model and document-centered design.

dynamically through embedded script. But the basic nature of Web pages and their interactive capabilities remain limited by the browser's request-response model and document-centered design.

For example, text (HTML) and images (JPEG, GIF, or PNG formats) are still the only media natively supported by Web clients; other media types, including audio, video, and vector graphics, require an external plug-in that launches in a separate window. The resulting experience tends to be confusing and discontinuous. As usability expert Jakob Nielsen has noted, "Although it's possible to provide Web-enabled applications through standard Web pages and the traditional browser interface, doing so neglects 20 years of GUI

advances and harkens back to the asynchronous interaction style last seen in the '70s on IBM 3270 terminals. It can be done — and it has been done — but it's not good for usability."¹

Indeed, the notion of a Web *page* itself connotes a degree of read-only finality that is better suited for publishing than for software interface design. If it were possible to re-envision the Web's presentation layer as an application interface, without its document-rendering baggage, user interface designers would certainly gravitate to a more dynamic metaphor from the get-go.

One approach might be to revisit the fat-client GUI, used extensively in the 1990s to build data access applications with Sybase PowerBuilder or Borland Delphi and seen today in such shrink-wrapped desktop applications as Microsoft Office. Fat clients offer users a rich, spontaneous environment capable of interacting over the network with server-based repositories.

As attractive as this description sounds, the fat-client approach is a poor fit for enhancing browser responsiveness for the following reasons:

- Fat clients have a large memory footprint, typically requiring tens of megabytes. This requirement is no problem for PCs, but it can be a significant barrier for handhelds and other resource-constrained devices — a profile of growing importance for Web software.
- Fat clients exploit platform-specific features, such as Microsoft's Win32 graphics device interface (GDI), to achieve both rich functionality and high performance. Fat-client GUI code therefore tends to be non-portable and must be recompiled and debugged for each new target. In an age of increasing client diversity, this requirement runs counter to best practices for developer productivity and code maintainability.
- Fat-client GUIs are typically coded in strongly typed programming languages, such as C++ or Java, or in specialized rapid application development (RAD) languages. It should be possible to leverage simpler JavaScript skills to develop Web front-ends.

Another approach to enriching the browser-based user experience is the style-sheet model, currently available in two industry-standard variants: Cascading Style Sheets (CSS) and Extensible Style Language Transformations (XSLT). Both these techniques centralize presentation design and decouple it from content, steps in the right direction for productivity and maintainability. Neither CSS nor XSLT, however, is concerned with enriching the browser's ability to deliver mixed media or to handle dynamic interactions.

Adding up the shortfalls in these approaches indicates the kind of features a superior rich Internet client would support:

- A lightweight (<500 KB), high-performance runtime for executing code and rendering multimedia content
- An extensible, event-driven object model for scripting interactivity
- The ability to access remote components and services hosted on standard application servers, including J2EE (Java 2 Enterprise Edition), Microsoft .Net, and SOAP-based Web services
- Support for multiple platforms and devices to maximize reach
- Zero install and administration on client, network, and firewall

Rich-Client Development with Macromedia Flash MX

Aberdeen has identified a handful of innovative products capable of meeting these requirements. Most, however, introduce new, unproven technologies on both client and server, offsetting their claimed benefits with hard-to-quantify risks. A notable exception is Macromedia Flash MX. To serve Flash content, a Web server

must be configured to recognize the SWF file format and invoke the client-side Flash Player, a lightweight (300 KB) runtime that renders multimedia content.

Flash is unique among rich-client runtimes in that the Flash Player is already installed on more than 400 million client devices, according to a recent independent study of online users. Flash also has the virtue of being demonstrably platform neutral, with a six-year track record of supporting such diverse platforms as Windows, Macintosh, Linux, Sun Solaris, MicrosoftTV, Symbian EPOC, PocketPC, Sony PlayStation 2, and others. Running in every significant browser configuration — from Internet Explorer 4 (5.0 on Mac OS) and Netscape Navigator 4 (4.5 on Mac OS) to AOL to Opera — Flash also eliminates the browser compatibility nightmare that plagues other rich-client implementations, such as DHTML and CSS.

Ubiquity, stability, and platform neutrality make Macromedia Flash MX a strong contender for deploying rich Internet front-ends. From a developer standpoint, Flash provides a highly productive and full-featured environment for assembling rich client-server applications. Web teams can create engaging, interactive user interfaces (UIs) using Flash's native scripting language, ActionScript, to control visual objects, handle events, and access remote data and logic over the network. An object-oriented language that strongly resembles standard ECMA-262 JavaScript, ActionScript is immediately accessible to today's Web teams using existing skill sets.

The Flash MX development environment resembles a cross between a rich graphics design tool and an object-oriented visual integrated development environment (IDE). Macromedia, justly reputed for its configurable design tools, has done a good job of integrating these fundamentally different feature sets. Creative designers can create beautiful animated text, graphics, and other rich media without tripping over code, while UI developers can focus on attaching ActionScript to these objects to implement interactivity. The result is a productive decoupling of workflows that gives distinct contributors their due.

In addition to its familiar programming paradigm and superior team workflow properties, Flash MX offers developers a radically streamlined means of connecting rich clients with data and logic living on applications servers. To implement client-server computing using earlier versions of Flash, developers needed to pass data as URL-encoded name/value pairs or, more recently, as serialized XML. The new version, Flash MX, offers a more seamless connectivity option, called Flash Remoting MX, which lets developers treat objects hosted on remote application servers as though they were local to the client. The Flash Remoting infrastructure takes care of data-type conversion, object lifetime management, serialization, and other details of client-server interactions — all of which developers must handle themselves when exchanging XML or other low-level I/O.

The combination of Flash MX, ActionScript, and Flash Remoting MX constitutes an attractive framework for assembling the next generation of responsive client-server

Internet applications. Before examining Macromedia's specific remoting implementation in detail, Aberdeen briefly reviews client-server connectivity in general and remoting as a relevant solution.

The Distributed Object Challenge

According to the Client/Server FAQ hosted by the Usenet group comp.client-server, "Client/server computing is the logical extension of modular programming." Just as dividing large blocks of software into modules results in better code productivity, quality, and maintainability, hosting the resulting modules on distinct machines (as opposed to a single monolithic processor) can, in principle, improve performance, manageability, and scalability. In practice, getting discrete software components to interact over a network — the distributed object challenge — is fraught with technical complexities.

A Java or C# class encapsulates data and the methods for manipulating it, and it is instantiated at runtime on a server for access by local processes. However, such objects cannot normally reach across language or address space boundaries. To accomplish such leaps, remote objects must be (1) accessible to clients by means of remote procedure calls (RPCs) and (2) capable of interpreting requests and bundling responses in a neutral wire protocol for messaging. Given these capabilities, clients do not need to know which language was used to build a server object or where on the network the object physically resides. They need to know only its name, its network address (e.g., as a Uniform Resource Identifier, or URI), and the interface the object supports.

Since the mid-1990s a number of distributed object architectures have appeared that all aim to provide platform-neutral client-server interactions across the Internet. A short list might include the following:

- J2EE
- Web server pages (e.g., ASP, JSP, ColdFusion, PHP)
- Microsoft .Net
- XML Web services

Each of these approaches has its virtues and liabilities, and all continue to enjoy widespread adoption. One thing that none of the frameworks listed can address, however, is the need to implement rich, responsive Internet applications using standard Web browsers or resource-constrained devices as clients. Java comes closest, with its platform-neutral AWT and Swing GUI classes, its applet model for browser deployment, and a special micro-edition, J2ME, for small devices. Java is certainly one to watch, but applets have a checkered past as client software, J2ME is immature and uncertain of widespread adoption, and Java as a programming language is overly complex for most Web applications.

What Web developers need is a rich-client authoring tool with scriptable access to the gamut of remote components and services. Microsoft's .Net aims to approximate these features on the Windows platform. ASP.NET supports scripting in either VBScript or JScript, and the .Net framework supports seamless XML connectivity to Web services and legacy COM components. Yet the authoring tool for .Net, Visual Studio, is a code-centric environment ideal for component developers, but heavy-handed for designers and UI builders. Microsoft's answer to rich-client development (other than fat Win32 clients) continues to be the browser-based alternatives DHTML and the DOM.

Taking Remote Control

Still, .Net gets many aspects of Internet client-server computing right, including its exemplary distributed object solution. Much of the press related to .Net centers on its support for XML Web services and its use of Simple Object Access Protocol (SOAP) messaging for inter-object communications. In fact, the .Net messaging model is based on the deeper concept of remoting, a superset of Web services that allow both SOAP- and binary-formatted communications. According to Microsoft, "one of the objectives of a remoting framework is to provide the necessary infrastructure that hides the complexities of calling methods on remote objects and returning results".²

The basic remoting architecture is shown in Figure 1. The remoting software has two parts, a client component and a server component (also called a *gateway*). When the client needs to access a remote service, the request is transformed by the client remoting component to a network-compatible RPC, including any parameters and input data. On the application server, the remoting gateway unbundles the request and passes it to the target service, which might be a Java Bean or other distributed object. The remoting gateway then returns result data to the client. The virtue of such a remoting framework is that client developers can treat remote objects as if they were local, instantiating them and calling their methods without explicitly managing their activation or lifetime on the application server. The local objects standing in for their remote counterparts are usually called *proxy objects*.

To summarize the requirements identified thus far, an ideal environment for developing rich Internet applications must combine rich-client authoring and scripted programmability with a remoting-style connectivity model.

Assembling Client-Server Solutions with Macromedia Flash Remoting MX

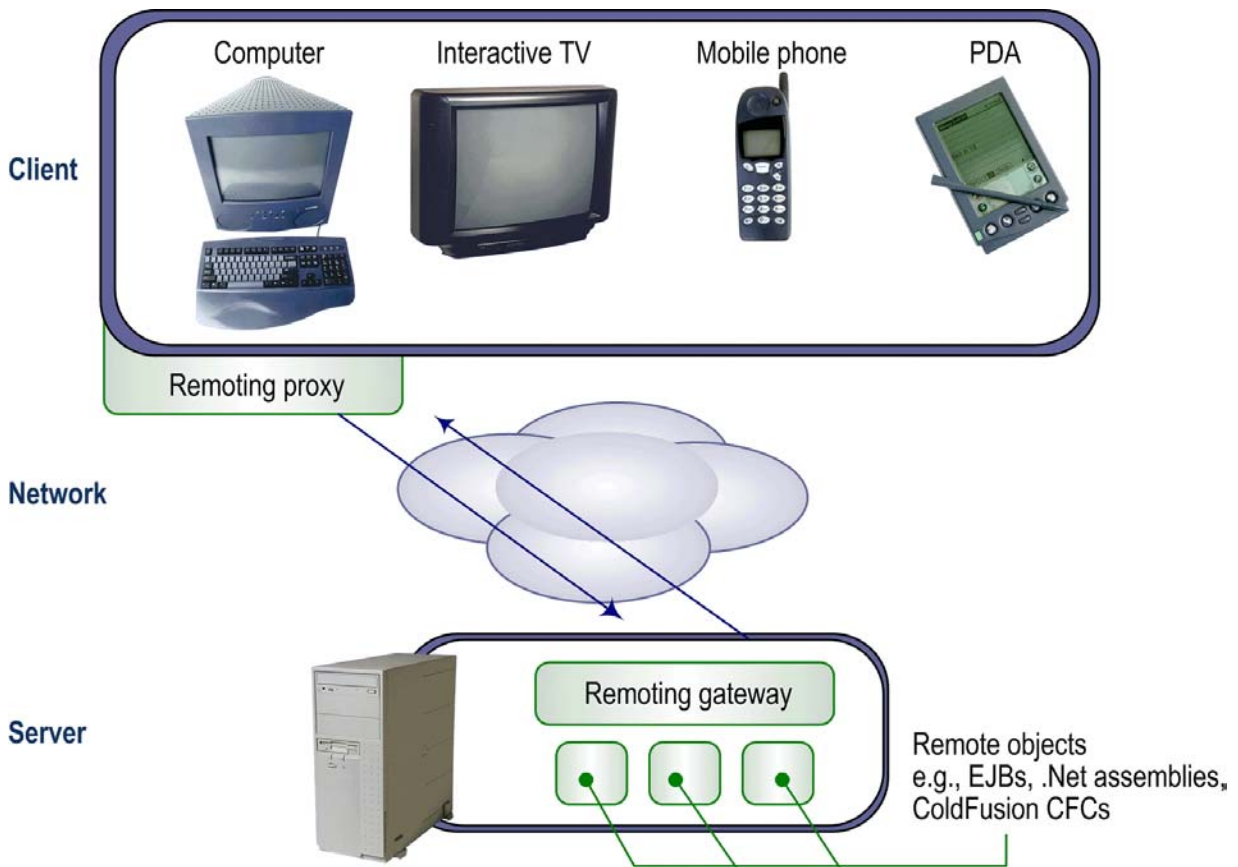
When it merged with Allaire Corp. in 2001, Macromedia acquired that company's deep server-side technical expertise, instantiated in the industry-leading application servers ColdFusion and JRun. Macromedia brought to the table its complementary roster of renowned client development tools, including Flash, Dreamweaver (the best-of-breed DHTML authoring environment), Fireworks (a Web

graphics editor), and FreeHand (a vector graphics application for creating print and Web illustrations).

The fruit of this client-server union is Macromedia's new MX strategy. MX is all about delivering great user experiences by enabling designers and developers to work together to build rich Internet applications. Web teams using the MX product suite will find that, for once, creative designers, UI developers, and component architects are being served by a toolmaker that "gets it" — not just one piece at a time, but end-to-end.

The October 2002 release of Flash Remoting MX is a case in point. Flash Remoting is an application server gateway that provides remoting-style connectivity between Flash clients and remote services. Through this gateway, ActionScript code in Flash movies can seamlessly invoke methods on remote services implemented anywhere on the network, in any relevant Web technology, such as the following:

Figure 1: Remoting Architecture



Source: Aberdeen Group, December 2002

- J2EE — EJBs, JavaBeans, and Plain Old Java Objects (POJOs), Java Management Extension (JMX) MBeans
- .Net services — ASP.NET technologies, ADO.NET data-binding adapters, .Net assemblies (DLLs), and .Net Web services
- Server pages — Microsoft ASP pages, Java Server Pages (JSP) and Servlets, Macromedia ColdFusion components and pages
- XML objects — SOAP-based Web services exposed through WSDL files, org.w3c.dom.Document objects, and other serialized XML objects

Flash Remoting MX bridges the gap between the rich-client development capabilities of Flash MX and the growing wealth of enterprise data and functionality residing on application servers.

To use Flash Remoting, a Web development team simply installs the Flash Remoting gateway on each target application server. The gateway installs as a servlet on J2EE servers, as a .Net assembly (flashgateway.dll) on Microsoft .Net servers, and as a native service in ColdFusion MX. In keeping with the goals of establishing a productive Internet client-server setup, Flash Remoting MX requires zero administration after install on either the application server or clients. In addition, the Flash Remoting gateway takes care of logging, error-handling, and security authentication, as well as automatically mapping the service request (RPC) to the appropriate object and server technology.

A Peek Under the Hood

Although Web designers and UI developers do not need to understand the intricacies behind Flash Remoting to benefit from it, readers of this paper may find it useful to peel back the curtain on just how much the technology accomplishes “automagically” on both the client and server to enable transparent service access.

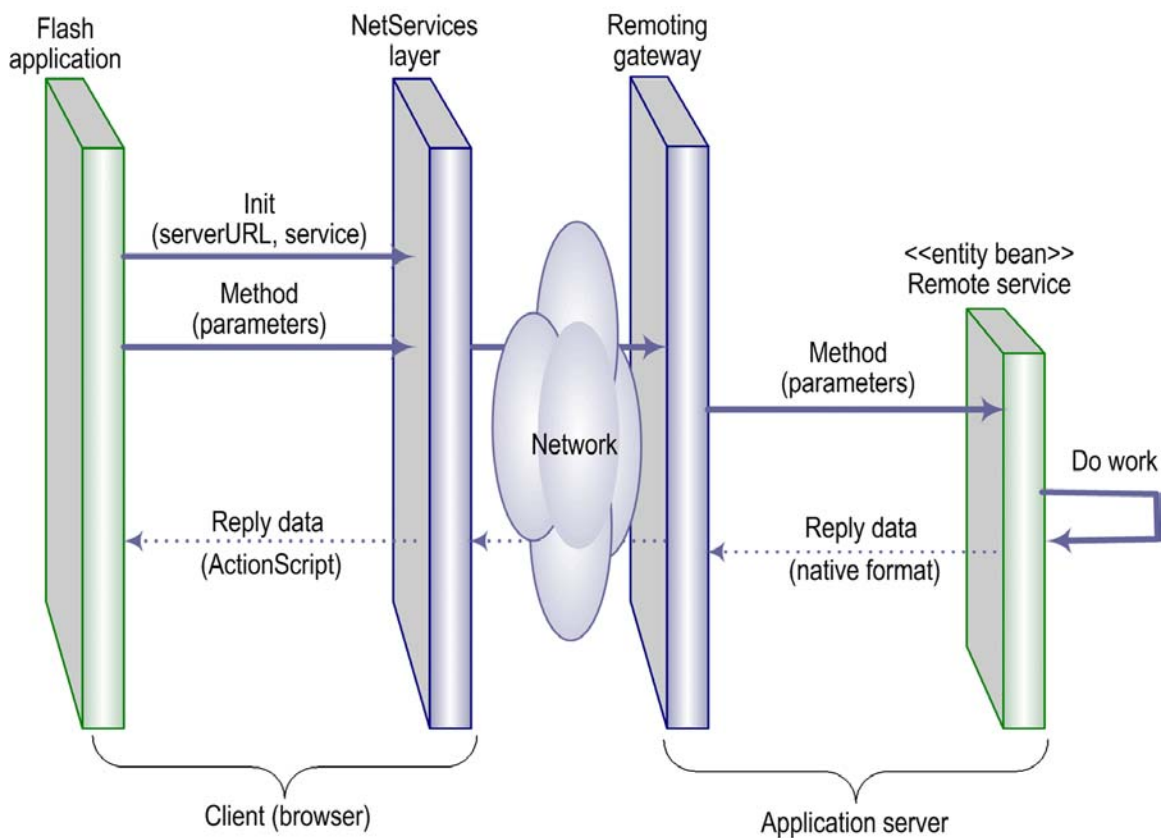
Figure 2 illustrates the basic Flash Remoting call sequence. A client Flash application is shown at left, and a remote service (here, a JavaBean) is shown at right. The remoting infrastructure consists of the NetServices layer, which resides on the client, and the remoting gateway, co-located with the target service on a remote server. Developers can instantiate and control this infrastructure through a straightforward application programming interface (API) comprising a set of new ActionScript functions built into Flash Player 6 (Release 40 and above). To access these NetServices functions, developers should simply include the file “NetServices.as” at the top of their scripts.

The NetServices layer and remote gateway are first initialized to establish a binary messaging channel and to create a local proxy object that represents the remote service. In the figure, this step is indicated by calling `init()`, which takes as arguments the gateway URL and a fully qualified service name. After initialization, Flash has access via the NetServices layer to the remote service’s public methods as

if they were local ActionScript resources. This convenience is worth emphasizing: besides testing `init()` for success and handling fault conditions, *developers do not need to write a single additional line of code* to manage communications with remote services. The data and methods published by the remote interface behave through the NetServices remoting bridge as if they were local ActionScript objects. Clearly, this is a major productivity boon for client-server developers.

Flash Remoting brings additional behind-the-scenes productivity benefits as well. For example, a complex series of transformations must usually be performed on exchanged objects to accommodate the substantial differences between clients and servers. With Flash Remoting, these transformations — mapping inter-system names, converting parameter and return value data types, serializing object payloads for transmission — are handled transparently by the client-side NetServices layer and server-side gateway. In other client-server solutions, developers are responsible for managing these and other nontrivial details of intersystem dialogues.

Figure 2: Flash Remoting Sequence Diagram



Source: Aberdeen Group, December 2002

With XML transfers, for example, remote objects must be serialized into DOM-style document trees, which developers must then write code to parse and load into local variables. Differences between data types in the remote service and client application, such as a Java Map populating an ActionScript associative array, must be overseen by the programmer, a potential source of hard-to-isolate bugs.

Of course, not every client-server installation should immediately switch from XML to Flash Remoting. Smaller projects in which XML is already in use may well serve current needs. It is important to note, though, that as projects grow over time and users require expanded access to back-end resources, a remoting infrastructure can streamline development and adaptation in ways XML (or its sibling, Web Services) cannot.

Raising the Speed Limit

In addition to enhancing developer productivity, Macromedia's offerings for implementing Flash Remoting bring important performance and scalability benefits. Three such benefits are discussed below: the reduction in network traffic associated with Flash Remoting's binary message format, the acceleration in object rendering resulting from Flash Remoting's close coupling to Flash and ActionScript, and the UI responsiveness enabled by Flash Remoting's asynchronous callback model.

To send messages to and receive results from remote services, Flash Remoting MX uses Action Message Format (AMF), a binary message format optimized for the ActionScript object model. Modeled on SOAP, AMF runs on top of HTTP and is therefore firewall safe and securable via HTTPS.

The choice of a binary format for client-server messaging streamlines communications considerably. Macromedia's preliminary investigation of AMF performance suggests that binary transfers reduce network traffic on the order of 50% over standard SOAP messaging. This finding accords well with similar findings from Microsoft comparing the performance of .Net remoting with binary-formatted versus SOAP-formatted HTTP channels.³ The bandwidth benefit is pleasing but not surprising: after all, binary data packs much more closely than verbose XML.

Reducing traffic will please network administrators, but the close coupling between AMF and ActionScript has another performance-boosting effect as well that directly affects the user experience. When serialized XML objects are exchanged between client and server, the payloads must not only be unwrapped from their SOAP envelopes, but also transformed to native client data types. This mapping slows the rendering process because data cannot be presented until it exists. Flash Remoting circumvents this transformation step altogether because AMF data types correspond 1:1 to ActionScript data types. As a result, Flash movies driven from remote services render noticeably faster than those driven via XML.

Perhaps the most visible impact from a user standpoint is the brisk responsiveness of applications driven through Flash Remoting MX. The experience is altogether counter to the stilted request-response model users have come to expect from the Web. Rather than forcing the client to wait for a remote service to reply, the NetServices layer returns control to the client immediately while the service request is being processed. When a reply is ready, NetServices notifies the UI through a callback (registered as an ActionScript event) and makes the reply data available to a client event handler. The process is smoothly asynchronous, enabling the creation of truly responsive Web-based user interfaces.

Division of Labor

There are also some interesting methodological benefits associated with Flash Remoting MX. Unlike other solutions to the distributed object challenge, which invariably require complex programming up and down the food chain, Flash Remoting divides the tasks involved in client-server development along natural team lines. For example, on the front-end, the presentation and information architecture aspects of Flash UI design can be allocated to multimedia creative artists. User interaction and event handling can be assigned to design technologists with specific ActionScript and DHTML skills. Implementation of server-side components — whether EJBs, .Net assemblies, or SOAP-based Web Services — can be handled by system programmers with specific object-building, transaction-processing, or multi-threading expertise.

Unlike other solutions to the distributed object challenge, Flash Remoting divides the tasks involved in client-server development along natural team lines.

This natural decomposition of software development workflows into self-consistent tasks is, as much as any technical advantage, one of Flash Remoting's most productive benefits. Furthermore, readers familiar with design patterns will recognize the close affinity between the tasks outlined above and the architecture prescribed by the so-called Model-View-Controller (MVC) pattern. Widely adopted by the software development community as a best practice for user interface-oriented applications, the MVC pattern contains the following elements:

- **Model** — Represents application data and logic. In a Web application, this typically consists of the application server tier(s) and database tier(s).
- **View** — Represents the user interface, consisting of user controls and information display.
- **Controller** — The controller represents event-handling logic that responds to user actions and messages the Model and View accordingly.

Revisiting the above task decomposition — identifying Flash movies with the View, event handling in ActionScript with the Controller, and server-side components with the Model — it is clear that Flash MX and remoting implement the MVC architecture in a natural way.

Aberdeen Conclusions

Macromedia Flash has come a long way from its origin as a plug-in for rendering animated vector graphics on the Web. Somewhat ironically, its success in this early role won the Flash Player its tenure on the vast majority of Internet clients, making Flash an obvious candidate for enriching Internet software. With the addition of a robust, familiar scripting language and XML connectivity, Flash was poised even before Macromedia introduced its client-server MX strategy to revolutionize user interface development on the Web.

With Flash MX in wide circulation and Flash Remoting MX available for all major application servers, Macromedia has given Web developers a productive framework for assembling scalable, secure, well-performing Internet applications.

Now, with Flash MX in wide circulation and Flash Remoting MX available for all major application servers, Macromedia has given Web developers a productive framework for assembling scalable, secure, well-performing Internet applications. In addition to providing a best-of-breed tool set for designing, coding, and debugging rich Web applications, Macromedia has paid careful attention to enabling teams to implement best-practice design patterns, such as Model-View-Controller. The Flash MX/Flash Remoting MX offering is a timely and fully realized answer to the distributed object challenge that provides significant development and runtime advantages over XML.

One risk area for teams new to Flash (or those focused primarily on its aesthetic power) is that they may be tempted to overindulge the tool's wealth of multimedia options. In Flash-specific research, Jakob Nielsen has found that contemporary Web users tend "to avoid anything that's overly hyped or promoted, especially if it looks like an advertisement".⁴ This "banner blindness" serves them well when surfing traditional Web sites, Nielsen notes, but it can cause users to miss key aspects of information architecture in Flash applications that misuse rich media. The cautionary upshot for rich interface designers is clear: Too much pizzazz can work against usability. Appropriately, Macromedia has begun educating its developer community to the do's and don'ts of user-centric Flash design.

With this caveat, Flash MX is a highly promising solution for creating rich Internet clients, and Flash Remoting MX is the best means for realizing Flash-based client-server applications. Together, these tools satisfy stakeholders' distinct needs along

the Web software development chain: designers, UI developers, server-side architects and system programmers, as well as system administrators and IT managers.

IT organizations tasked with developing the next generation of rich intranet, extranet, and Internet applications should fully explore Flash Remoting MX as a productive solution to building responsive client-server systems.

Notes

¹ Jakob Nielsen, "Flash and Web-Based Applications," "Alertbox," Nielsen Norman Group, November 25, 2002. <www.useit.com/alertbox/20021125.html>

² Piet Obermeyer and Jonathan Hawkins, "Microsoft .Net Remoting: A Technical Overview," MSDN Library, July 2001. <msdn.microsoft.com/library/en-us/dndotnet/html/hawkremoting.asp>

³ Piet Obermeyer and Jonathan Hawkins, "Performance Comparison: .NET Remoting vs. ASP.NET Web Services," MSDN Library, September 2002. <<http://msdn.microsoft.com/library/en-us/dnbda/html/bdadotnetarch14.asp>>

⁴ Jakob Nielsen, "Flash and Web-Based Applications," "Alertbox," Nielsen Norman Group, November 25, 2002. <www.useit.com/alertbox/20021125.html>

To provide us with your feedback on this research, please go to www.aberdeen.com/feedback.

*Aberdeen Group, Inc.
260 Franklin Street, Suite 1700
Boston, Massachusetts
02110-3112
USA*

*Telephone: 617 723 7890
Fax: 617 723 7897
www.aberdeen.com*

*© 2002 Aberdeen Group, Inc.
All rights reserved
December 2002*

Aberdeen Group is a computer and communications research and consulting organization closely monitoring enterprise-user needs, technological changes and market developments.

Based on a comprehensive analytical framework, Aberdeen provides fresh insights into the future of computing and networking and the implications for users and the industry.

Aberdeen Group performs projects for a select group of domestic and international clients requiring strategic and tactical advice and hard answers on how to manage computer and communications technology. This document is the result of research performed by Aberdeen Group. It was underwritten by Macromedia, Inc. Aberdeen Group believes its findings are objective and represent the best analysis available at the time of publication.