

# A “NEXT GENERATION” ARCHITECTURE FOR HTTP

**WILLIAM C. JANSSEN, JR.**  
*Xerox Palo Alto Research Center*

The notion of a “next generation” HTTP arose from studying the Internet congestion problem caused by the Web. In 1995, Simon Spero presented the original design for what he called HTTP-NG.<sup>1</sup> It was a multiplexing protocol that allowed a single TCP connection to handle multiple fetches and to multiplex the returning data. In early 1997, a Xerox/Microsoft/Digital group submitted an HTTP-NG project proposal to the World Wide Web Consortium.<sup>2</sup> The plan was to study current Web usage characteristics and create a prototype for a new protocol based on a distributed-object framework, which was intended to flesh out the original distributed-object notions of HTTP 0.9. The usage study would drive test scenarios for comparing the new protocol with the existing HTTP. The W3C approved this proposal, and the activity began in June 1997.

First and foremost among the design requirements was that the Web continue to work on top of HTTP-NG as well as it works on HTTP. This meant that the protocol had to support all current functions of browsers and servers. It also meant that new applications should be easily deployable and old applications easily extensible.

Another key design goal was to improve performance. Therefore, HTTP-NG had to use available bandwidth more efficiently and take protocol latencies into consideration. TCP was to remain the basis of the connection architecture; but the protocol set should work on top of any reliable byte stream layer, and new TCP connections should be created only when absolutely necessary to avoid congestion problems and connection setup overhead.

Finally, the new architecture should support Java RMI, CORBA, and DCOM applications without the need for tunneling messages of these communication protocols through HTTP POST calls. The union of these systems

HTTP-NG is a comprehensive rethinking of the Web's underlying protocol. It provides a framework for defining new Web applications, a powerful messaging system, and a multiplexing transport layer.

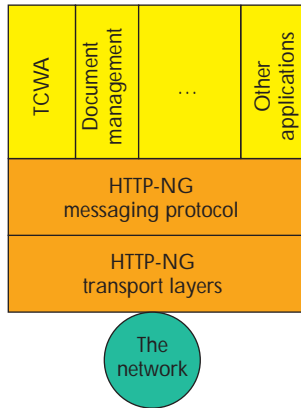


Figure 1. The HTTP-NG three-layer architecture: transport layer, messaging protocol layer, and application layer, which includes “The Classic Web Application.”

was adopted as the basis for what a state-of-the-practice distributed-object framework must provide for application deployment.

We began by investigating the architectures of a wide variety of existing systems, from TCP to the X Window System protocol, and from various RPC and distributed-object systems to the Web protocols themselves. From HTTP 1.1<sup>3</sup> we drew a characterization of the current Web as an application protocol. We also studied HTTP-based systems like Web Distributed Authoring and Versioning (WebDAV), from which we drew ideas of what features other application protocols could use.

### THREE-LAYER ARCHITECTURE

The HTTP-NG architecture is a three-layer system, as shown in Figure 1. The layers are derived from the HTTP 1.1 specification. At the top of the stack is an application layer; in the middle is a messaging protocol layer; and at the bottom, a transport layer. The interface between the application layer and the messaging

layer is known as the “messaging interface,” and the interface between the messaging layer and the transport layer is known as the “transport interface.”

### The Application Layer

The application defined in HTTP 1.1 is the familiar one of fetching Web pages, handling proxies and caches, and submitting forms for use by CGI-bin scripts. It uses the methods GET, HEAD, POST, and PUT, defined in Web “resource” objects. In HTTP-NG, this application is abstracted from the details of the wire protocol into what we call “The Classic Web Application.” TCWA was submitted to IETF as an Internet draft in August 1998.<sup>4</sup> HTTP-NG also provides a system for defining methods used for other applications, such as WebDAV. The system uses a mechanism for typing resources to determine which methods are appropriate for delivery to which resources. A Web resource that supports TCWA, for example, is said to support the TCWA object type, and receives only the methods specified by the TCWA API.

The mechanism for defining new methods and grouping them into object types supports both static and dynamic typing, which is often useful where a parameter’s type must be extended in the future, or where a parameter can be of a broad variety of types. The actual type of a dynamically typed parameter can be determined at runtime. In addition, a method extension technique allows variants of a single method to be related so that a method signature can evolve over the lifetime of an application without disturbing already deployed implementations.

The type system incorporates and unifies the type systems of

the Java RMI, CORBA, and DCOM protocols, so that APIs expressed in any of these systems can be directly mapped to HTTP-NG without tunneling. It provides “functional compatibility” with the other type systems. For instance, in the CORBA type system, any value of an “interface type” can be either a real value or nil, the empty value. In HTTP-NG, every value of a “remote object type”—the element that corresponds to CORBA’s interface type—must be a real value. However, the HTTP-NG remote object type can be used to construct a reference type that has the same behavior as the CORBA interface type. Thus, HTTP-NG provides the same functionality through a different mechanism.

### The Messaging Layer

The messaging system describes a standard way of marshaling values of any type expressible in the application layer’s type system. The marshaling rules are straightforward and, unlike HTTP 1.1, independent of the invoked method or the currently supported application. They are based on the IETF’s External Data Representation. XDR is already an IETF draft standard,<sup>5</sup> and widely implemented in distributed systems. Since the parameters and results for all methods are expressed in the type system, the complexity of marshaling a request or reply message does not increase regardless of the number of applications or the number of methods defined in any application.

The messaging layer is primarily a request-response messaging protocol, just like HTTP 1.1. However, it improves on the performance characteristics of HTTP 1.1 in several ways. For example, it uses caching and binary message

formats to reduce the number of bytes actually transferred as well as the amount of CPU time needed to marshal and unmarshal them. The HTTP-NG messaging layer uses additional control messages to further reduce bandwidth requirements by establishing sessionwide characteristics, rather than requiring attributes to be sent with each message. Another performance improvement comes from exploiting the application layer's type system to remove the interdependency of individual headers and to reduce the implementation complexity.

The messaging format, currently called w3ng, exploits session caching as used in systems like DCE RPC, the X Window System wire protocol, and Java RMI. For example, the first time a method or resource is invoked over a connection, the full identifier is transferred in the request message, and both sides of the connection cache it. On subsequent method invocations, a small connection-specific integer value is transferred instead. Furthermore, these caches can be loaded by both ends of a connection upon establishment, which is particularly useful for standard applications such as TCWA.

The target resource for a method invocation is passed relative to the previous target, using a technique like the one used for relative URLs.<sup>6</sup> Instead of passing the entire resource identifier, only the part of the identifier that differs from the last one is actually passed over the connection. The receiver reconstructs the full identifier by applying that difference to the previously passed identifier.

The techniques used in the messaging layer can reduce overhead significantly compared to

## THE HTTP-NG ACTIVITY

The HTTP-NG project has a home page, which also includes information about the project's mailing list. This page can be found at <http://www.w3.org/Protocols/HTTP-NG/>.

A more complete description of the architecture is available at <http://www.w3.org/TR/WD-HTTP-NG-architecture/>.

The specification for the w3ng wire protocol is at <http://www.w3.org/TR/WD-HTTP-NG-wire/>.

The specification for the WebMUX transport protocol is at <http://www.w3.org/TR/WD-mux/>.

The open source modifications to tcpdump are available at <ftp://ftp.parc.xerox.com/transient/janssen/tcpdump-3.4a6-rpc.tar.Z>.

## PARC'S ILU

The Inter-Language Unification system (ILU) is a multilanguage object interface system from Xerox PARC. The object interfaces provided by ILU hide implementation distinctions between different languages, different address spaces, and operating system types.

ILU can be used to build multilingual object-oriented libraries ("class libraries") with well-specified language-independent interfaces, with library components implemented in a wide variety of programming languages, including Java, C++, Python, and Common Lisp. It can also be used to implement distributed systems or to define and document interfaces between the modules of nondistributed programs. ILU interfaces can be specified in either the OMG's CORBA Interface Definition Language, or ILU's Interface Specification Language (ISL). ILU can be used as a CORBA ORB, or to implement object-oriented Web browsers or servers. ILU is available as open source from <ftp://ftp.parc.xerox.com/pub/ilu/ilu.html>.

similar existing protocols. Tests with CORBA's IIOP 1.0, for example, have shown message header sizes up to seven times larger than obtained with HTTP-NG for the same message. In absolute terms, a common usage such as interrogating a server to see whether a Web resource has changed since some specified time, for instance, might take only 12 bytes for the request and 12 bytes for the response, using w3ng and WebMUX (described below).

The messaging format for HTTP-NG is binary—unlike

the text-based format used in HTTP 1.1. The binary format improves performance in two ways. First, nonstring values need not be formatted into strings on the sending side and decoded from strings on the receiving side. Instead, values can often be copied directly to and from their memory representations using efficient operations such as `memcpy`. Second, the binary forms of such values tend to use fewer bytes than a string form of the value. An integer value of 10,000,000, for instance, uses just four bytes in

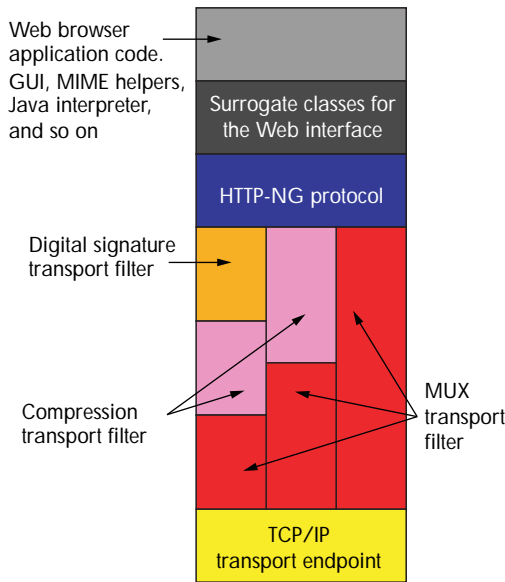


Figure 2. Architecture of an HTTP-NG Web browser.

HTTP-NG, but eight or more bytes in HTTP 1.1.

**The Transport Layer**

The general notion of a stack or stream of filters—each modifying the data flowing through it in some way—has been used in many systems. Perhaps the best-known example is how applications can be composed in “pipes” to create new applications in Unix. Similarly, HTTP-NG supports application-specified composition of “transport stacks,” which can manipulate the messages passed through them.

The transport layer allows for a stack of transport elements that can be arranged in various orders. It provides for multiplexing of multiple virtual connections over a single underlying reliable byte stream such as TCP/IP, and it provides for bidirectional use of a single byte stream so that servers can make callbacks to clients without opening additional byte streams.

It also provides record-marking of message boundaries on an underlying byte stream, necessary for supporting higher level message-based systems. It allows messages to be passed in “chunks” or fragments, which allows both multiplexing of simultaneous messages, and rapid dispatching of higher value data often found near the start of a data segment, such as the size information at the beginning of a GIF file.

A particularly useful combination of transport elements, called the WebMUX stack, has been defined for HTTP-NG. It is designed to fit directly under the marshaling layer and directly above the TCP/IP transport element, but it can also be used in conjunction with other transport elements such as compression or security elements.

The WebMUX layer does the record-marking and supports fragmentation of messages. It provides for optional flow-con-

trol of transmissions, so receivers can control the data rate of the sender. It also supports multiplexing of up to 252 virtual connections over a single underlying connection. Each of these virtual connections can have different transport stacks above the WebMUX layer, allowing, for example, multiple security or caching contexts to be used simultaneously. In addition, the WebMUX layer provides endpoint identification to both sides of the underlying connection, which facilitates using the same connection for callbacks from server to client. Since the way to pass endpoint identification is well specified, firewalls can observe and intervene in the interaction if necessary.

**PROTOTYPING RESULTS**

The revised HTTP-NG architecture and protocol have been published as Internet Drafts and implemented in Xerox PARC’s ILU object-oriented prototyping framework. Early benchmarking of the relatively unoptimized ILU-based implementation against the heavily optimized HTTP 1.1 libwww browser library and Apache server have already demonstrated HTTP-NG’s performance improvements.

For example, fetching the W3C Microscape benchmark (a single Web page with several embedded images) with HTTP-NG took an average of 0.19 seconds on the PARC testbed, versus 0.23 seconds for HTTP 1.1. Pipelined HTTP 1.1 requires the transfer of almost 7,000 more bytes than HTTP-NG for that same benchmark. Performance against other messaging protocols is even more striking: CORBA’s IIOP 1.0 messaging format on the Microscape

benchmark, for instance, took an average of 0.58 seconds. We intend to continue benchmarking activities at PARC, and to publish all the code and configuration information used for our testing.

HTTP-NG functionality has been added to several heavily used Web applications, including experimental versions of the Apache Web server and Internet Explorer browser (see Figure 2). PARC has produced a version of the standard IP traffic monitoring tool, tcpdump, that is capable of interpreting and displaying HTTP-NG messages. The ILU implementation of HTTP-NG and the modifications to tcpdump are freely available as open source systems. (See the sidebar, "The HTTP-NG Activity," for more information.)

The next step in the W3C process is to take the architecture and protocols developed thus far to the IETF for further review and revision. To that end, we gave IETF Birds-of-a-Feather sessions at the August 1998 IETF meeting in Chicago, and at the December 1998 meeting in Orlando. We expect the IETF to approach the continued design of HTTP-NG in stages, probably beginning with a working group in the Transport Area to standardize the WebMUX transport architecture. ■

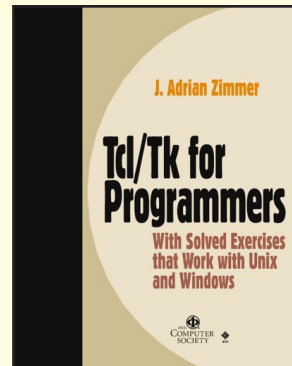
#### ACKNOWLEDGMENTS

Many people have contributed to the development of this architecture, including Mike Spreitzer, Dan Larner, Jim Gettys, Henrik Frystyk Nielsen, Jim Pitkow, Paul Bennett, Daniel Veillard, Paul Leach, and Andrew Begel. In addition, most of the good ideas in HTTP-NG have been drawn from successful features of previous systems designed by a cast of thousands.

**William C. Janssen, Jr.**, is a research staff member at the Information Systems and Technologies Lab at Xerox's Palo Alto Research Center. He is a principal of the PARC Inter-Language Unification project, and he leads the PARC HTTP-NG project. His research interests include object interface systems, distributed object systems, programming language design, and group coordination systems.

#### REFERENCES

1. S. Spero, *Next Generation Hypertext Transfer Protocol*, white paper, 1995; available online at <http://sunsite.unc.edu/ses/ng-notes.txt>.
2. W. Janssen et al., *Briefing Package for HTTP-NG Project*, 1997; available online at <http://www.w3.org/Protocols/HTTP-NG/Group/Brief.html> to members of the World Wide Web Consortium.
3. R. Fielding et al., *Hypertext Transfer Protocol — HTTP 1.1*, IETF Request For Comments, 1997; available online at <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2068.txt>.
4. D. Larner, *HTTP-NG Web Interfaces*, IETF Internet Draft, 1998; available online: <http://info.internet.isi.edu/in-drafts/files/draft-larner-nginterfaces-00.txt>.
5. R. Srinivasan, *XDR: External Data Representation Standard*, IETF Request for Comments, 1995; available online at <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1832.txt>.
6. R. Fielding, *Relative Uniform Resource Locators*, IETF Request for Comments, 1995; available online at <http://info.internet.isi.edu:80/in-notes/rfc/files/rfc1808.txt>.



## Tcl/Tk for Programmers

With Solved Exercises that Work with Unix and Windows

by J Adrian Zimmer

This introduction to Tcl/Tk bridges the gaps between introductions, comprehensive manuals, and collections of scripts that solve particular problems. There are over 200 exercises with solutions for both Unix and Windows platforms.

*Tcl/Tk for Programmers* introduces high-level Tcl/Tk scripting language to experienced programmers with either Unix or Windows backgrounds. It includes a short introduction to TCP/IP, introductions on writing client-side scripts and GUI interfaces as well as integrating scripts with C/C++. In addition to covering version 8.0/8.0, the book describes the major differences between versions 8.0/8.0, 7.6/4.2, and the experimental alpha version 8.1/8.1. Zimmer has extensive knowledge of Tcl/Tk programming and currently runs a consulting and training company based on his experience.

560 pages. 7" x 10" Softcover.  
October 1998. ISBN 0-8186-8515-8.  
Catalog # BP08515  
\$35.00 Members / \$45.00 List

Online Catalog

<http://computer.org>

+1.800.CS.BOOKS • +1.714.821.8380

