

THE STATELESS
NATURE OF HTTP
CONTINUES TO BE A
PROBLEM FOR
APPLICATIONS ON
THE WEB.

THIS ARTICLE
SHOWS HOW TO
RECURSIVELY PRESERVE
STATE DURING HTTP
SESSIONS USING A
SIMPLE CGI PROGRAM.

DYNAMIC ARGUMENT EMBEDDING: *PRESERVING STATE ON THE WORLD WIDE WEB*

ARUN IYENGAR • IBM T.J. WATSON RESEARCH CENTER

At the IBM T.J. Watson Research Center we implemented a way of preserving state during HTTP sessions by modifying hypertext links to encode state information. We call the method dynamic argument embedding, and it was developed in response to problems we encountered implementing the Coyote Virtual Store, a transaction-processing system prototype.

Virtual store applications, such as IBM's Net.Commerce and Netscape's Merchant System, typically need to maintain information such as the contents of shopping baskets while customers are shopping. We wanted our application to be flexible enough to maintain such state information without restricting the sorts of HTML pages a customer might view. We also wanted a system that didn't require extensions to the hypertext transfer protocol (HTTP) and so could be implemented on any standard Web server and client browser. Finally, we wanted to permit customers to access several accounts at once by using the browser's cache to concurrently store pages corresponding to multiple invocations of the virtual store application.

The two most common methods of preserving state on the Internet are HTML forms and Netscape cookies. The former limits client-server request-response interactions to dynamically generated forms, and the latter currently works only with browsers and servers that support cookies. Neither approach could support our need for concurrent access to multiple accounts.

Our method uses a server program that filters HTML pages before they are sent to the client and encodes state information within hypertext links. In essence, it allows the server to recapture state

from the modified hypertext link passed back from the client, and in this way provides persistent state information across a series of HTTP requests. Dynamic argument embedding therefore offers one way to overcome the stateless nature of HTTP sessions without changing the underlying nature of the protocol, and it has the flexibility we needed for our application.

PRESERVING STATE WITH HTTP

HTTP communications between a client and server are stateless. This means that every request from the client to the server is treated independently.¹⁻⁴ Information from previous connections is not maintained for use in future connections.

There are times, however, when it is essential to maintain state information from previous client-server interactions. All current methods of doing so represent state information by assigning values to variables known as state variables. These are then passed to every server program that might need them. The challenge lies in propagating the state variables to the server programs.

HTML forms

One common method for handling state on the Web involves the use of HTML (hypertext markup language) forms.²⁻⁴ Servers respond to client requests with dynamically generated HTML forms that contain the state information embedded in hidden variables. These HTML forms are typically created by common gateway interface (CGI) programs. The hidden variables are passed back to the server after the client submits the form.

To illustrate, let's suppose that a business transaction server is communicating with a client. The client identifies itself to the server so that access to the proper account is established. Once this initial identification is made, the server should maintain some state information so the client doesn't have to identify itself for every transaction. With HTML forms, the server responds to each client request with a dynamically generated form that contains the user ID as a hidden argument to the form (Figure 1). When the client submits each form, the server identifies the client from the hidden user ID argument. The state information is thus passed back and forth between the client and server (Figure 1).

This approach limits the types of interactions that are possible between a client and a server while state is being preserved. The server must always respond to the client with a dynamically generated HTML form containing hidden variables, and there is no way to preserve state, for example, when the client is browsing static HTML files.

Cookies

Netscape Communications invented cookies to preserve state without the limitation imposed by HTML forms.⁵ At present cookies require nonstandard extensions to both

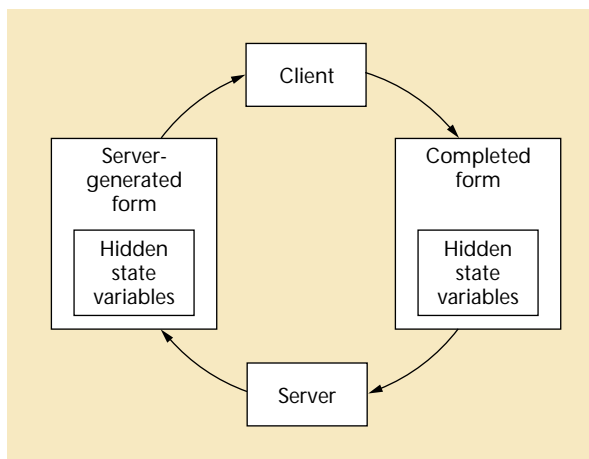


Figure 1. A commonly used approach for preserving state on the World Wide Web is to pass the state variables as hidden arguments to HTML forms.

servers and browsers. A server can satisfy an HTTP request by appending a cookie to its response. The cookie contains a description of the range of URLs for which the state is valid. The cookie is stored by the client browser, and any future HTTP requests made by the client for one of the URLs specified in the cookie will include a transmittal from the client to the server of the state object stored in the cookie.

Servers introduce cookies to clients by including a Set-Cookie header as part of the response to a request. Typically, cookies are created by CGI programs. For example, a browser might receive the following cookie from a server:

```
Set-Cookie: USERID=TIGER; path=/dir1;
           expires=Wednesday, 09-Nov-99 23:12:40 GMT
```

This cookie contains the value for a variable `USERID`. Whenever the client requests a URL from the server under the path `/dir1`, it sends the cookie:

```
Cookie: USERID=TIGER
```

In this example, the cookie expires at 11:12:40 p.m. on November 9, 1999. If no expiration date and time are given, a cookie expires when the browser session terminates.

DYNAMIC ARGUMENT EMBEDDING

Dynamic argument embedding is a general technique for maintaining state that associates state with *conversations*. A conversation is a sequence of communications between a client and one or more servers in which the client selects the next page by following a hypertext link provided by a server.

Dynamic argument embedding modifies hypertext links to encode state information by rewriting the link to invoke a spe-

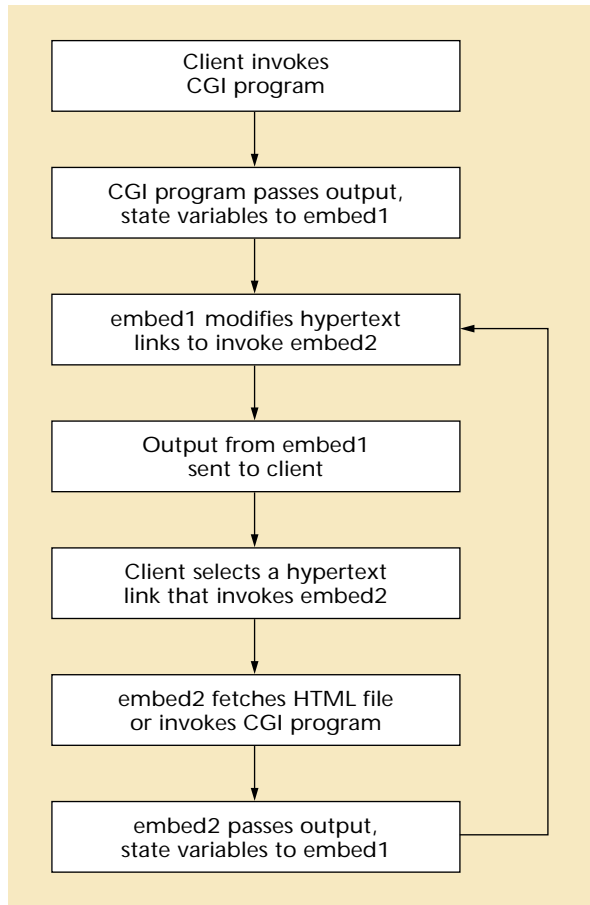


Figure 2. Steps in dynamic argument embedding.

cial program known as an *argument embedder*, which passes the state variables to all CGI programs. A client following a conversation in which state is preserved using dynamic argument embedding will be passed hypertext links that are all calls to the argument embedder.

The algorithm for preserving state is depicted as a flow chart in Figure 2. A client invokes a CGI program on a server. The program determines that state variables should be embedded in the conversation so that all CGI programs that could be invoked from the conversation will be given access.

The CGI program generates an HTML page with hypertext links for the client to continue the conversation. However, instead of returning the HTML page to the client unmodified, the CGI program has been adapted to invoke the embed1 module of the argument embedder by passing it the HTML page and all of the state arguments.

The embed1 module modifies all hypertext links in the HTML page to be calls to the embed2 module of the argument embedder (Figure 3). If a link was to an HTML file, embed2 is passed an absolute reference to the file and all state arguments. If a link was to a CGI program, embed2 is

passed an absolute reference to the CGI program, the original arguments to the CGI program, a parameter delimiting the end of the original arguments, and the state arguments.

For example, suppose that the state variables were $x = 32$ and $y = 45$. A link to an HTML file

```
<a href="http://www.watson.ibm.com/mail.html">
```

would be modified to

```
<a href="http://www.watson.ibm.com/
cgi-bin/embed2?url=//www.watson.ibm.com/
mail.html&x=32&y=45">
```

A link to a CGI program

```
<a href="http://www.watson.ibm.com/
cgi-bin/prog?arg1=55">
```

would be modified to

```
<a href="http://www.watson.ibm.com/
cgi-bin/embed2?url=//www.watson.ibm.com/
cgi-bin/prog&arg1=55&comma=1&x=32&y=45">
```

(The string "comma=1" allows embed2 to distinguish the original argument arg1 from the state variables x and y . If there is a danger of "comma" conflicting with another variable of the same name, a more sophisticated method for this could be used.)

The embed1 module now sends the modified HTML page to the client. All hypertext links in the modified page are calls to embed2. The client selects a hypertext link to continue the conversation.

Two possibilities arise. One is that the first argument to embed2 is a file, and the remaining arguments are state variables. If so, the file is fetched and passed, along with all of the state arguments, to embed1. The process returns to the third step in Figure 2. For example, the transformed HTML hypertext link from Figure 3 would cause embed2 to fetch the file mail.html and pass the HTML text from the file along with the state variables $x = 32$ and $y = 45$ to embed1.

The second possibility is that the first argument to embed2 is a CGI program. The remaining arguments are the original arguments to the CGI program, a separator argument, and the state variables. The CGI program is invoked on the original arguments and the state variables. The resulting output is passed along with all of the state variables to embed1. The process returns to the third step in Figure 2. For example, the transformed CGI hypertext link from Figure 3 would cause embed2 to invoke prog on the arguments arg1=55, $x = 32$, and $y = 45$. The output from prog is passed along with the state variables $x = 32$ and $y = 45$ to embed1.

In some cases, it is desirable to pass state to a CGI program but not to recursively embed state in the output generated by the program, such as when the CGI program is already preserving state in its output using dynamic argument embedding. A modification to Step 3 handles this situation. The `embed1` module of the argument embedder recognizes this class of CGI programs by a special string in the program name and appends the state arguments to the already existing arguments. For example, suppose that the state variables again were $x = 32$ and $y = 45$ and the special string was "5668." A hypertext link to the CGI program

```
<a href="http://www.watson.ibm.com/cgi-bin/prog5668?arg1=55">
```

would be modified to

```
<a href="http://www.watson.ibm.com/cgi-bin/prog5668?arg1=55&x=32&y=45">
```

An application can prevent state variables from being propagated to a hypertext link by placing a comment `<!--NO_STATE-->` before the hypertext link.

COMPARING THREE METHODS OF PRESERVING STATE

What are some of the advantages and disadvantages of current methods of preserving state? Let's take a closer look at how HTML forms, cookies, and dynamic argument embedding compare. First, on the question of compatibility: HTML forms can maintain state only when all responses from the server are dynamically generated forms. This limitation compromises the usefulness of this method for a large class of applications. In contrast, dynamic argument embedding and cookies can preserve state for applications that return static, or generate dynamic, HTML pages of any kind. In terms of compatibility with the HTTP protocol, HTML forms and dynamic argument embedding work with any browser and server that support HTTP. Cookies are presently nonstandard and require special support.

Lifetimes of state variables

By default, cookies are valid until the end of a browsing session. The Web application that creates a cookie must specify an expiration date and time to override default behavior. This can be a problem, as it is often not possible to predict how long users will be viewing HTML pages. If the expiration date is too soon, the state information can be invalidated in the middle of an application; if it is too far in the future, cookies with state infor-

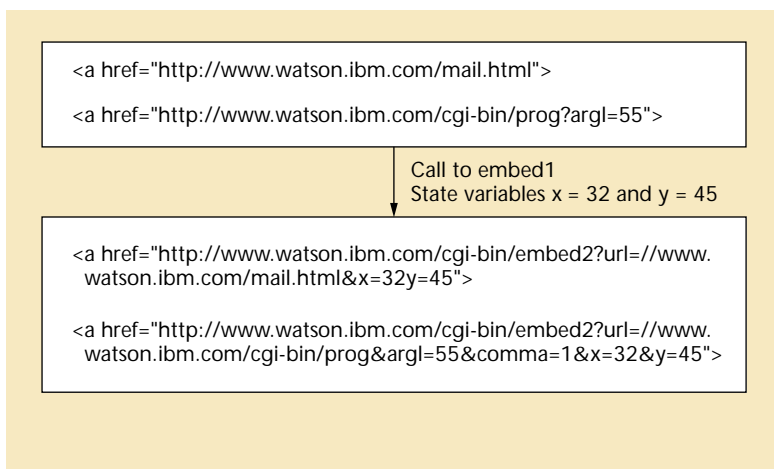


Figure 3. Hypertext links are modified in this fashion by the `embed1` module of the argument embedder.

mation will persist and could confuse other applications (or the same application if it is restarted without invalidating the cookies). As a rule, confidential state information should not have a lifetime beyond that of the application that might access it.

State variables maintained by dynamic argument embedding or HTML forms are an integral part of the Web application, and therefore expiration dates and times do not have to be specified. The state variables will not expire in the middle of an application or remain after the application has stopped executing. Nor do dynamic argument embedding and HTML forms depend on the correctness of system clocks as do cookies.

On the other hand, there are situations when it is desirable to have state variables expire while a browser is still viewing pages corresponding to the application. While cookies handle this situation naturally, dynamic argument embedding and HTML forms can be used only if expiration is handled at the application level. There are also situations where state variables should be preserved after an application has finished executing. For these situations, only cookies can be used.

Performance

Cookies entail little overhead for the server and result in the best performance of the three techniques. The primary overhead incurred by both dynamic argument embedding and HTML forms results from invoking CGI programs to create each page, since both techniques require all pages to be dynamically generated by the server. CGI is typically implemented by forking off a separate process that terminates after the CGI program is finished, a mechanism that entails a high overhead.⁸

We measured the performance of dynamic argument embedding and HTML forms using the Silicon Graphics WebStone* benchmark. WebStone tests maximum server

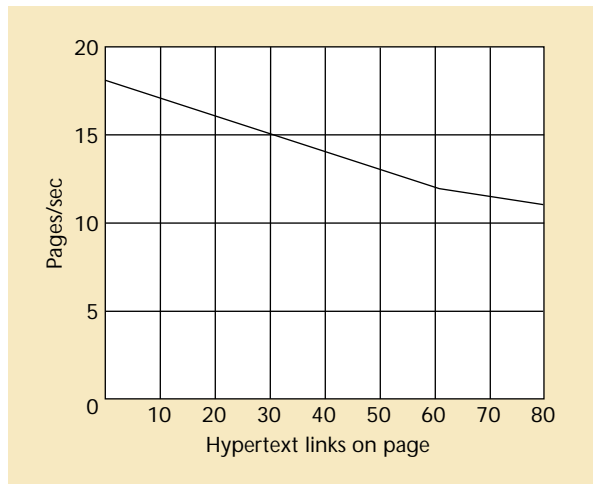


Figure 4. The performance of dynamic argument embedding varies with page size. Performance is primarily affected by the number of hypertext links per page.

throughput by simulating multiple clients and issuing as many requests as possible to the server. We used version 1.9 of the IBM Internet Connection Secure Server running on an IBM Powerstation 580. CPU processing power was the limiting resource. The network had sufficient bandwidth to handle all requests without bottlenecking the system. All requests were for text files. Results are presented in Figure 4.

Our files ranged from small files with no hypertext links to large ones containing up to 80 links. For workloads consisting entirely of static files, the server was able to deliver 54 to 63 files per second before the CPU became 100 percent utilized. For a workload consisting entirely of trivial CGI programs, the server could deliver 18 dynamic pages per second before the CPU became 100 percent utilized. The CPU delivered approximately 18 dynamic pages per second for CGI programs that used either HTML forms or dynamic argument embedding when pages were small and contained fewer than 10 hypertext links. In these situations, overhead is due almost entirely to CGI and not to work done by the argument embedder.

As we moved to larger pages with more hypertext links, the performance of dynamic argument embedding degraded slightly. In the worst case when the workload consisted entirely of large pages containing 80 hypertext links, the server could deliver only 11 dynamic pages per second. Sixty percent of the overhead resulted from CGI and 40 percent from the dynamic argument embedder. This case is, however, extreme and unlikely to occur in practice.

The bottom line for both dynamic argument embedding and HTML forms is that performance is primarily hindered by the overhead of invoking CGI for server programs. Any technique that reduces this overhead will improve the performance of both techniques.

There are a number of promising techniques for accomplishing this that allow clients to execute server programs without spawning separate processes each time. One approach is to link server programs directly with the Web server; another is to prefork multiple processes or threads, which the Web server communicates with to invoke server programs. The Netscape Server API (NSAPI) and the Internet Server API (ISAPI) by Microsoft use the first approach, while FastCGI by Open Market uses the second.⁹⁻¹¹ Over time, more servers will support such APIs and the overhead of invoking server programs will drop substantially. As this happens, the overhead for dynamic argument embedding and HTML forms will diminish.

Many applications that maintain state are already producing a high percentage of pages dynamically via CGI programs. Dynamic argument embedding and HTML forms should not significantly affect the performance of these applications.

Security

Cookies, HTML forms, and dynamic argument embedding all support Netscape's* Secure Sockets Layer (SSL) so that state variables can be encrypted as they are sent across a network.⁶ HTML forms and dynamic argument embedding also support Enterprise Integration Technologies'* Secure HTTP (S-HTTP).⁷

With HTML forms, each dynamic form can pass state variables to only one URL, and when needed, the application program can ensure that state variables are transmitted securely to that URL. With dynamic argument embedding and cookies, users can often continue conversations by selecting from a variety of hypertext links. Both dynamic argument embedding and cookies have capabilities to prevent the unencrypted transmission of specific state variables under such circumstances. Dynamic argument embedding can also convert HTTP requests automatically to SSL requests to protect state variables.

Dynamic argument embedding implements its security features by allowing you to name state variables with a SECURE or SECUREFORCE suffix. Variables that have the SECURE suffix are not embedded in any URL where the protocol is not https (SSL) or shttp (S-HTTP). Variables with the SECUREFORCE suffix cause the dynamic argument embedder to modify hypertext links as necessary to ensure secure transmission. This may change some hypertext links from the http protocol to https.

The previously mentioned `<!--NO_STATE-->` comment can be used to prevent state variables from being transmitted to hypertext links specifying the unencrypted HTTP protocol. Adding `<!--SECUREFORCE-->` before such a link changes the protocol to https if any state variables are embedded in the link.

It is possible for cookies to be passed to a malicious URL that is not part of the application. HTML forms and

dynamic argument embedding pass state variables only to hypertext links that are part of the application. An application can also receive a cookie created by an unrelated application; if the application is expecting a cookie with the same name, incorrect behavior could result.

Because cookies are stored on disk, they can also cause problems if the disk memory on the client system is accessible to malicious parties. HTML forms and dynamic argument embedding can be trusted in this situation only if disk caching at the client is turned off and the client does not store HTML pages with confidential information on disk.

Re-entrant Web applications

With dynamic argument embedding, browsers can cache multiple copies of pages corresponding to the same application with different instantiations of state variables.

To illustrate what this means in practice, consider a Web application that maintains information about users. Each invocation maintains a state variable representing the account ID. With dynamic argument embedding, a user who has multiple accounts can access two or more accounts concurrently by using the browser's cache and flipping between pages corresponding to the different accounts. This is possible because the different invocations will have different sets of URLs, which can all be cached.

Cookies and HTML forms do not allow a single browser to work concurrently on multiple invocations of the program, since the account information from the most recent invocation would overwrite any previous invocation.

EXTENSIONS TO DYNAMIC ARGUMENT EMBEDDING

Dynamic argument embedding can be generalized to handle any situation that requires HTML pages to be modified dynamically during a conversation. HTML pages obtained both locally and from remote servers could be modified. For example, the embed1 module could be adapted to filter out objectionable groups of words and hypertext links. The result would be a censoring device for Internet conversations.

In another example, suppose a client is in a conversation where the names of major corporations appear frequently in the text. The client contacts a server that has access to a database of home page URLs for major corporations. The server wishes to add hypertext links each time the name of a company in the database appears in an HTML page. This can be accomplished by modifying the embed1 module to search HTML pages for all company names in the database. Whenever such a name is found, a hypertext link to the company's home page would be inserted into the HTML text returned to the client.

A key feature of dynamic argument embedding is the argument embedder's access to pages before they are

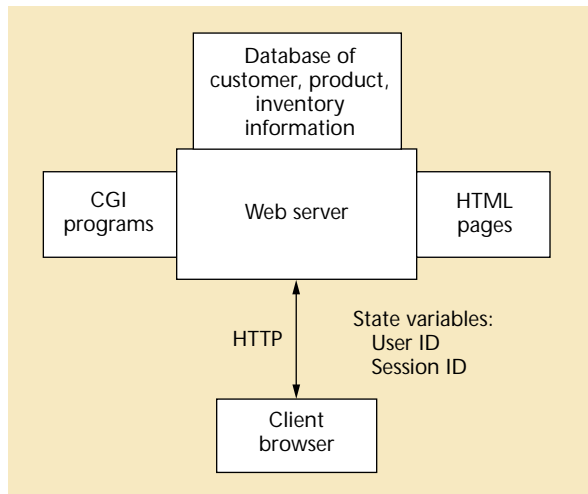


Figure 5. The Coyote Virtual Store architecture.

sent to the client. A variety of possible uses follow from this feature. As one example, let's consider proxies that sit between the client browser and the network to filter and analyze incoming Web pages. One difficulty in implementing such proxies is insertion between the client and the network. In addition to taking time to install, specialized proxies might be needed for different client and network configurations.

Using our technology, a server application—let's call it a Web site analyzer—can be written to implement proxies for any standard browser connected to the Web with no need for special hardware or software. A user would simply access the Web site analyzer URL and be prompted for the base URL of a site to be analyzed. The analyzer would analyze each page accessed by hypertext links before sending it to the client. The Web site analyzer could provide options for combining information from several conversations.

IMPLEMENTATION

We have implemented dynamic argument embedding for the Coyote Virtual Store, a server-side software prototype developed at the IBM T.J. Watson Research Center that allows businesses to sell goods over the Internet. The virtual store maintains information about the customer, inventory, and products in a database. It assumes very little about the format of its product catalogs. Catalogs may consist of HTML files as well as CGI programs. They may exist on local or remote servers. The architecture for the Coyote Virtual Store is shown in Figure 5.

To purchase products, update customer information, or access sensitive information, the client must authenticate

*URLs from pages 53-55

Netscape Communications • www.netscape.com
Enterprise Integration Technologies • www.eit.com
WebStone • www.sgi.com/Products/WebFORCE/WebStone

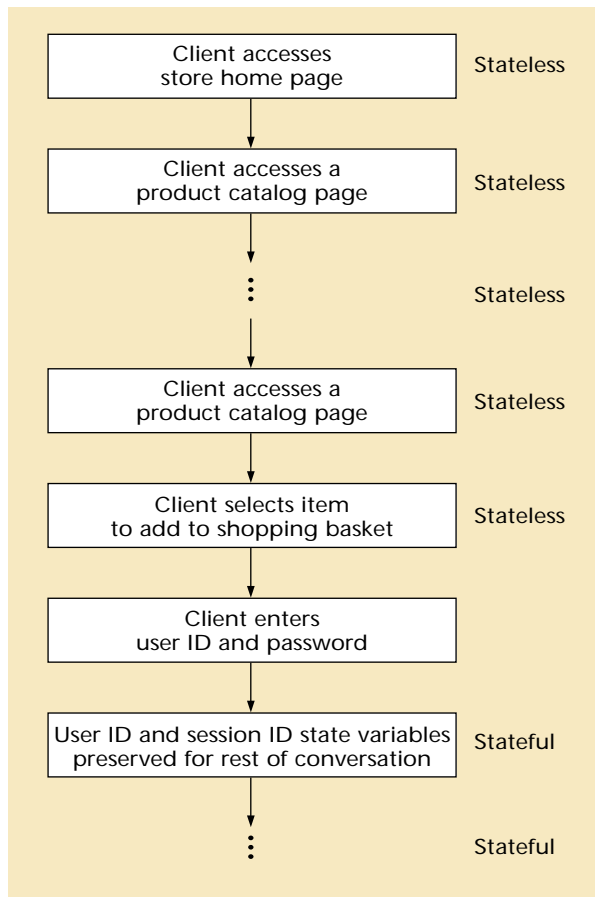


Figure 6. The Coyote Virtual Store transaction processing system embeds user ID and session ID state variables into conversations.

itself by entering a user ID and password. (New clients can pick their user ID and password, and are asked to provide additional information such as postal address and phone number.) Once authenticated, the user ID is embedded into the conversation as a state variable using dynamic argument embedding. Multiple transactions can now be performed without reauthentication.

Authentication also creates a session ID that is passed between the client and server as a state variable. Session IDs are randomly chosen for each conversation over a large enough set so that they cannot be guessed. Each time the user performs a secure transaction by invoking a CGI program, the server makes sure a valid session ID was passed to the CGI program along with the user ID. SSL prevents other users from obtaining the session ID and making unauthorized transactions. The advantage of using session IDs rather than the client's password as a state variable for authentication is that passwords are thereby given extra protection by being transmitted across the network only once during the initial authentication. In addition, passwords will not be substrings of the HTML pages passed to the client.

A typical conversation with a Coyote Virtual Store would begin with the client accessing the store's home page. From there, the user would begin to browse offerings from the product catalog. At this point, communication is stateless, and the goal is to defer authentication so that the widest possible audience will view the catalog. At some point, the user finds an item to add to a shopping basket. The user must then enter a user ID and password to continue. A session ID is created and embedded into the conversation as a state variable along with the user ID. The user can now view additional products, modify the shopping basket, commit to purchases, or view and update database information without reauthentication (Figure 6).

CONCLUSION

Dynamic argument embedding preserves state by embedding state information in hypertext links of HTML pages. The algorithm can be generalized to other applications where it is necessary to modify or examine Web pages before they are displayed by the client's browser. Dynamic argument embedding has the general advantage of being a more standard solution to the problem of maintaining state because the arguments themselves become part of the application and are not independently implemented. ■

REFERENCES

1. Internet Engineering Task Force, *HTTP: A Protocol for Networked Information*, www.w3.org/pub/WWW/Protocols/HTTP/HTTP2.html.
2. N.J. Yeager and R.E. McGrath, *Web Server Technology*, Morgan Kaufmann Publishers, San Francisco, 1996.
3. I.S. Graham, *The HTML Sourcebook*, John Wiley, New York, 1995.
4. J. December and M. Ginsburg, *HTML and CGI Unleashed*, Sams.net, Indianapolis, Ind., 1995.
5. Netscape Communications Corp., *Client Side State—HTTP Cookies*, www.netscape.com/newsref/std/cookie_spec.html.
6. Netscape Communications Corp., *The SSL Protocol*, www.netscape.com/newsref/std/SSL.html.
7. Enterprise Integration Technologies, *Secure HTTP*, www.eit.com/creations/s-http/.
8. See the following Web resources for CGI: NCSA, *The Common Gateway Interface*, hohoo.ncsa.uiuc.edu:80/cgi/overview.html; Yahoo, www.yahoo.com/Computers/World_Wide_Web/CGI_Common_Gateway_Interface/; The Web Developer's Virtual Library, www.stars.com/; and W3C, www.w3.org/pub/WWW/CGI/.
9. *Netscape Server API*, www.netscape.com/newsref/std/server_api.html.
10. *Microsoft Internet Server API*, www.microsoft.com/msdn/sdk/platforms/doc/sdk/internet/src/isapimg.htm.
11. *FastCGI*, www.fastcgi.com/.

Arun Iyengar is a research staff member at the IBM T.J. Watson Research Center. His research interests include the World Wide Web, electronic commerce, and parallel processing. Iyengar received a PhD and MS in computer science from the Massachusetts Institute of Technology and a BA in chemistry from the University of Pennsylvania.

Readers may contact Iyengar at IBM Research Division, T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, aruni@watson.ibm.com.