Preserving State on the World Wide Web Using Dynamic Argument Embedding

Arun Iyengar
IBM Research Division
T. J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Abstract

The HTTP protocol which is used for communicating over the World Wide Web is stateless; every request from a client to a server is treated independently. We have developed a new method for preserving state on the World Wide Web known as dynamic argument embedding. The technique has advantages over HTML forms and Netscape cookies, which are two commonly used state preservation techniques. Unlike Netscape cookies, dynamic argument embedding works with all browsers and servers supporting the HTTP protocol. Our method can be generalized to other situations where it is necessary to dynamically modify or monitor HTML pages before they are received by a browser.

Index Terms- World Wide Web, state, dynamic argument embedding, cookies, HTML forms, HTTP.

1 Introduction

The Hypertext Transfer Protocol (HTTP) [5, 11, 4, 2] is stateless. Every request from a client to a server is treated independently. The server doesn't maintain information from previous connections for future connections. However, there are situations where it is essential to maintain state information from previous interactions between a client and a server. This paper presents a new method called *dynamic argument embedding* for maintaining state information over the World Wide Web which has key advantages over existing methods.

In our method and in existing methods, state information is represented by assigning values to variables known as *state variables.* The state variables are then passed to every server program (i.e. CGI script) which might need them. The difficulty lies in propagating the state variables to CGI scripts.

One commonly used method for handling state on the Web involves the use of HTML forms [4, 2, 11]. Servers respond to client requests with dynamically generated HTML forms which contain the state embedded in hidden variables of the form. These HTML forms are typically created by CGI scripts. The hidden variables are passed back to the server after the client submits the form. For example, suppose that a business transaction server is communicating with a client. The client needs to identify itself to the server in order to access the proper account. After the client identifies itself, the server needs to maintain some state information so that the client doesn't have to keep identifying itself for every transaction. The server responds to each request with a dynamically generated form which contains the userid as a hidden argument to the form. When the client submits each form, the server identifies the client from the hidden userid argument. The state information is thus passed back and forth between the client and server (Figure 1).

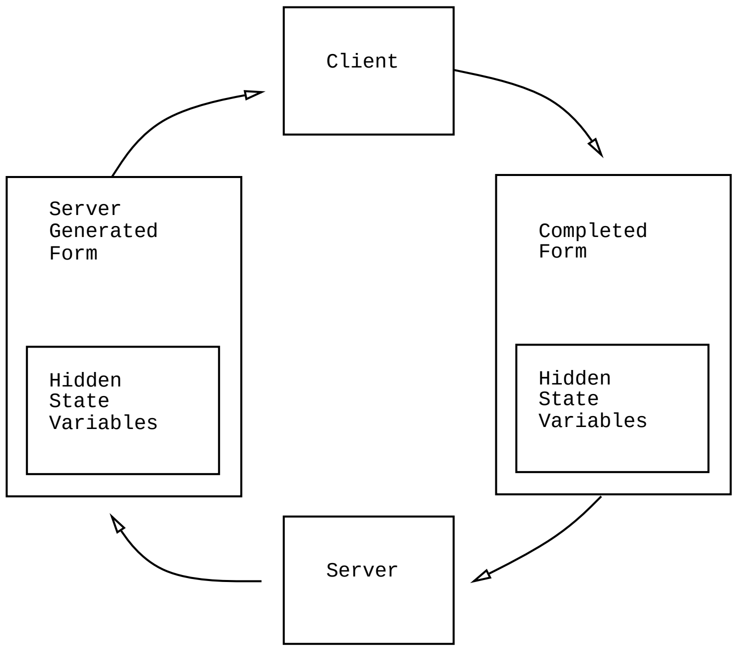


Figure 1: A commonly used approach for preserving state on the World Wide Web is to pass the state variables as hidden arguments to HTML forms.

The problem with the approach just outlined is that it limits the types of interactions 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. There is no way to preserve state while the client browses HTML files. For example, suppose that the client wishes to browse a catalog in the middle of the session. The catalog consists of HTML files. There is no way to allow the client to browse different HTML files in the catalog without losing the state information.

The limitations imposed by forms make them impractical for many applications. Netscape Communications Corporation has invented *cookies* which preserve state without the limitations of HTML forms [7]. Cookies require nonstandard extensions to both 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 URL's for which the state is valid. The cookie is stored by the browser running on the client. Any future HTTP requests made by the client for one of the URL's specified in the cookie will include a transmittal of the state object stored in the cookie from the client to the server.

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 scripts. For example, a browser might receive the following cookie from server *s*:

```
Set-Cookie: USERID=ARUNI; 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 server *s* under the path */dir1,* it sends the cookie:

```
Cookie: USERID=ARUNI
```

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

However, there are limitations to Netscape cookies. Dynamic argument embedding has a number of advantages over cookies which are summarized here. A more detailed comparison is given in Section 2.2.

- Cookies are a nonstandard feature and are not supported by all browsers and servers. By contrast, dynamic argument embedding works with all clients and servers which support the HTTP protocol.

- Most browsers which support cookies allow users to reject them. Applications which depend on cookies for state preservation will not work properly if the user is not accepting cookies. By contrast, we are not aware of any browser which can disable dynamic argument embedding.

- Many users dislike cookies because they leave a persistent record of URL's which have been visited on disk and can be used by unknown parties to track Web sites that have been visited by a client. Dynamic argument embedding doesn't have either of these features.

- By default, cookies are valid until the end of a browsing session. The Web application which creates a cookie must specify an expiration date and time in order to override default behavior. This mechanism creates problems because it is generally not possible to predict how long users will be viewing HTML pages corresponding to a Web application which utilizes cookies. If the expiration date is too soon, the state information will be invalidated in the middle of the application. If the expiration date is too far in the future, cookies with stale information will remain which could confuse other applications or the same application if it is restarted without invalidating the cookies. Furthermore, confidential state information should never have a lifetime beyond that of the application which might access it. Long lifetimes for such data introduce security risks.

By contrast, state variables in dynamic argument embedding are an integral part of the Web application. There is no way for them to expire in the middle of the application or remain after the application has stopped executing.

- Dynamic argument embedding allows browsers to simultaneously cache multiple instantiations of the same CGI script or HTML file with different state variables in situations where it is not possible to do so using cookies. This is accomplished by encoding the state variables within the URL.

- Dynamic argument embedding provides precise control over which URL's receive state variables. Cookies provide a method for restricting the URL's where a cookie can be sent which is not as precise as our method.

- Dynamic argument embedding provides a wider variety of options for controlling security in transmitting state variables.

2 Dynamic Argument Embedding

Dynamic argument embedding [6] is a general technique for maintaining state on the World Wide Web which doesn't require any extensions to the HTTP protocol. It associates state with *conversations.* A conversation is a sequence of communications between a client and one or more servers in which the client always selects the next page by following a hypertext link provided by a server.

More formally, a series of HTML pages $h_1, h_2, ..., h_n$ constitutes a conversation if:

1. $h_1, h_2, ..., h_n$ were all viewed by a client, and

2. for all i such that $1 < i \leq n$, page h_i was obtained by following a hypertext link on page h_{i-1}.

In an uninterrupted conversation, the client simply follows $n - 1$ hypertext links to get from page h_1 to h_n without ever backtracking. In an interrupted conversation, the client backtracks at least once. By backtracking, we mean that the client:

1. Initially visits a page h_i where $1 \leq i < n$,

2. Views other pages either by following hypertext links, explicitly accessing URL's, using the browser's cache, etc.

3. Returns to page h_i by reloading h_i from memory (such as from the browser's cache).

Dynamic argument embedding modifies hypertext links to encode state information. Hypertext links are changed to invoke a special program known as an *argument embedder*. The argument embedder passes the state variables to all CGI scripts. It also modifies hypertext links to invoke the argument embedder. A client following a conversation in which state is preserved using dynamic argument embedding will be passed hypertext links which are all calls to the argument embedder.

State is preserved in the following manner:

1. A client invokes a CGI script p_1 on a server.

2. CGI script p_1 determines that state variables $x_1, x_2, ..., x_n$ should be embedded in the conversation so that all CGI scripts which could be invoked from the conversation are given access. CGI script p_1 generates an HTML page h with hypertext links for the client to continue the conversation. Instead of returning h to the client unmodified, p_1 has been adapted to invoke the *embed1* module of the argument embedder by passing it h and all of the state arguments:

$$embed1(h, x_1, x_2, ..., x_n)$$

3. The *embed1* module modifies all hypertext links in h to be calls to the *embed2* module of the argument embedder. 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 script, *embed2* is passed an absolute reference to the CGI script, the original arguments to the CGI script, 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 script

```
<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 could be used to pick a unique variable name instead of *comma*.

4. The *embed1* module sends the modified HTML page h' to the client. All hypertext links in h' are calls to *embed2*.

5. The client selects a hypertext link to continue the conversation. The hypertext link is a CGI call to *embed2*. There are two possibilities:

   (a) 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 Step 3. For example, the hypertext link

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

   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*.

   (b) The first argument to *embed2* is a CGI script. The remaining arguments are the original arguments to the CGI script, a separator argument, and the state variables. The CGI script 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 Step 3. For example, the hypertext link

   ```
   <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">
   ```

   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*.

The algorithm is depicted pictorially in Figure 2. Figure 3 shows the manner in which hypertext links are modified.

In some cases, it is desirable to pass state to a CGI script but not to recursively embed state in the output generated by the CGI script. An example would be a situation where the CGI script 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 scripts by a special string in the script name. Whenever such a script is encountered, *embed1* appends the state arguments to the already existing arguments. For example, suppose that the state variables were $x = 32$ and $y = 45$. A hypertext link to the CGI script

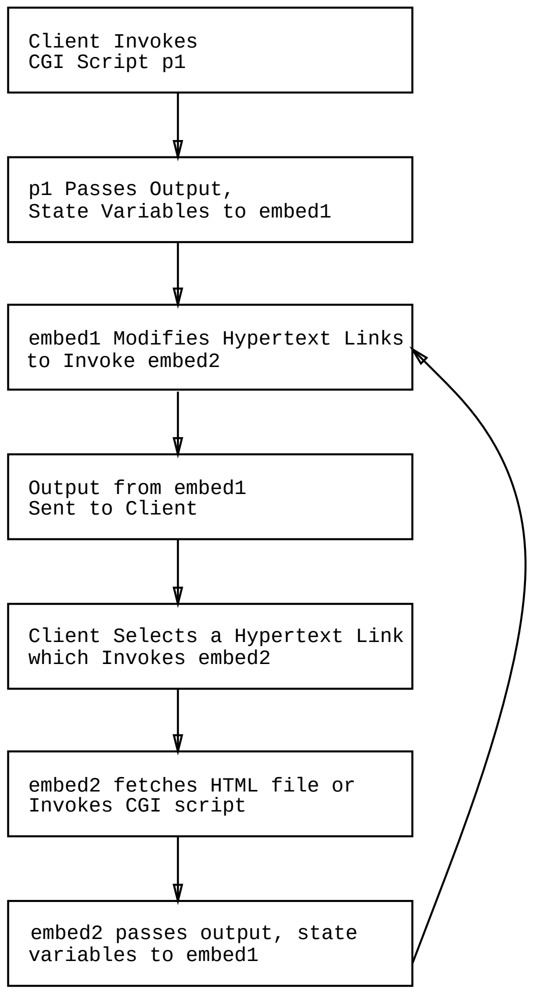
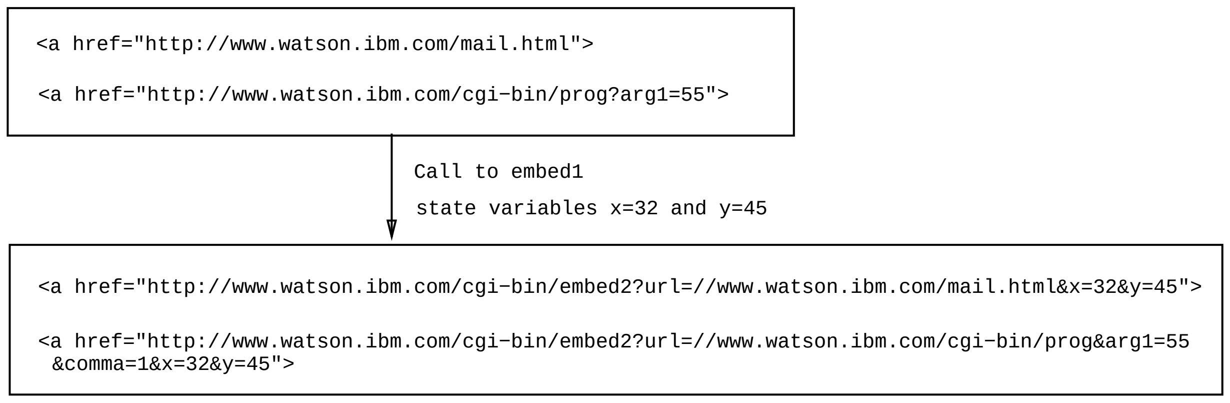

Client Invokes CGI Script p1
p1 Passes Output, State Variables to embed1
embed1 Modifies Hypertext Links to Invoke embed2
Output from embed1 Sent to Client
Client Selects a Hypertext Link which Invokes embed2
embed2 fetches HTML file or Invokes CGI script
embed2 passes output, state variables to embed1

Figure 2: The major steps in dynamic argument embedding.



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

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

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

<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">
```

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

```
<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.

Security

Clients and servers often need to communicate securely over the World Wide Web in order to prevent third parties from obtaining confidential information. Secure Sockets Layer (SSL) [9] by Netscape is a commonly used encryption system providing secure communication over the Internet. Secure HTTP (S-HTTP) [3] by Enterprise Integration Technologies is another well known encryption system which is not used as frequently as SSL. Both SSL and S-HTTP can be used in conjunction with dynamic argument embedding to transmit state variables securely.

Dynamic argument embedding has features for enforcing that state variables are transmitted securely. These features are only needed when the application doesn't use a secure protocol for all hypertext links which might embed state variables. One such feature is to name state variables with a *SECURE* or *SECUREFORCE* suffix. For example, a variable *x_SECURE* will only be embedded in hypertext links which are passed securely. The dynamic argument embedder recognizes hypertext links corresponding to secure communications via SSL or S-HTTP by the protocol substring of the URL. SSL uses *https* for the protocol substring while S-HTTP uses *shttp*.

A state variable *x_SECURE* causes the dynamic argument embedder to modify hypertext links as necessary to ensure that *x_SECURE* is transmitted securely. This may change some hypertext links from the *http* protocol to the *https* protocol corresponding to SSL.

The difference between a *SECURE* and *SECUREFORCE* variable is that *SECURE* variables are not sent to servers if the corresponding URL specifies the unencrypted *http* protocol. By contrast, *SECUREFORCE* variables are always sent to servers regardless of the protocol specified in the original URL. If this protocol is insecure, it is converted to a secure one.

Hypertext links which specify the unencrypted *http* protocol can also have directives associated with them which prevent state variables from being transmitted insecurely. The previously mentioned

```
<!-- NO_STATE -->
```

comment prevents any state variables from being embedded within the hypertext link. A comment of the form

```
<!-- SECUREFORCE -->
```

before a hypertext link specifying the unencrypted *http* protocol changes the protocol to the *https* protocol corresponding to SSL if any state variables are embedded within the hypertext link.

Other Variations

Using dynamic argument embedding, state is passed back and forth between the client and server. In order to limit the amount of state which is passed back and forth between the client and server, it is possible to store most of the state in a file system or a database residing on the server. In this case, it is only necessary to pass an index back and forth between the client and server. Using a language such as Java [1], it is also possible to store state in a file system or database residing on the client.

Java could also be used to download the argument embedder from the server onto the client. The argument embedder would then run on the client, who would then no longer have to go through the server to preserve state. An advantage to this approach is that the load on the server is reduced. In addition, the client will be able to continue conversations even if the server from which the client obtained the applet goes down or becomes unavailable due to a network failure.

2.1 Performance

One drawback to dynamic argument embedding is that it can increase the load on the server. Every Web page which is accessed must be scanned by the dynamic argument embedder which is implemented as a CGI script. Our performance measurements indicate that the primary overhead results from invoking the CGI script and not from the actual work done by the CGI script. We have measured similar overheads for a wide variety of CGI scripts. On most platforms, each CGI script is executed by forking off a separate process which terminates after the CGI script is finished. This mechanism entails a high overhead.

Any technique which reduces the overhead of the CGI interface will improve the performance of dynamic argument embedding. There are a number of promising techniques for reducing this overhead which are likely to become commonplace in the near future. The basic approach is to allow clients to execute server programs without spawning separate processes each time. This can be accomplished by linking server programs directly with the Web server, preforking multiple processes or threads which the Web server communicates with to invoke server programs, and the use of multithreaded Web servers. FastCGI [10] by Open Market uses the first approach while the Netscape Server API (NSAPI) [8] and ISAPI by Microsoft use the second approach.

Over time, more servers will support API's such as Fast CGI, NSAPI, and ISAPI, so the overhead of invoking server programs will drop substantially. If the cost of invoking server programs is not high, dynamic argument embedding will not substantially hurt performance.

Another way to reduce load on the server is to use Java to run the dynamic argument embedder on the client.

Finally, it should be noted that a significant percentage of applications which maintain state produce a high percentage of pages dynamically via CGI scripts. For these classes of applications, dynamic argument embedding should not affect performance significantly.

2.2 A Comparison of Dynamic Argument Embedding, Netscape Cookies, and HTML Forms

HTML forms are not sufficient for maintaining state in many Web applications because state can only be maintained while all responses from the server are dynamically generated forms. By contrast, dynamic argument embedding and Netscape cookies allow state to be preserved for applications which generate both static and dynamic HTML pages of any kind.

Dynamic argument embedding has a number of advantages over cookies. Cookies are nonstandard and only work with browsers and servers with special support. By contrast, dynamic argument embedding works with any browsers and servers which support the HTTP protocol.

Even if a browser supports cookies, an application relying on cookies will not function properly if the browser is configured to reject cookies. By contrast, we are not aware of any browser with the ability to disable dynamic argument embedding.

Many users dislike cookies because they leave a persistent record of URL's which have been visited on disk and can be used by unknown parties to track Web sites that have been visited by a client. These are two major reasons why users configure browsers to reject cookies. Dynamic argument embedding doesn't have either of these features.

A cookie can have an expiration date and time which are explicitly specified at the time of the cookie's creation. If an expiration date and time are not specified, the cookie expires when the browser's session ends. This mechanism for specifying cookie lifetimes presents problems because it is usually not possible to predict the optimal expiration time of a cookie. If the expiration time is too soon, the state information will be invalidated in the middle of an application. If the expiration date is too far in the future, cookies with stale information will remain which could confuse other applications or the same application if it is restarted without invalidating the cookies. Furthermore, security could be compromised if cookies containing confidential information are allowed to remain valid beyond the point where they are needed. Another problem with this approach is that it relies on the correctness of system clocks. If either client or server clocks are incorrect, cookies may fail to work properly.

By contrast, state variables maintained by dynamic argument embedding are an integral part of the Web application. There is no need to specify an expiration date and time for the state variables. They will not expire in the middle of an application or remain after the application has stopped executing. In addition, dynamic argument embedding does not depend on the correctness of system clocks.

With dynamic argument embedding, browsers can cache multiple copies of pages corresponding to the same application with different instantiations of state variables. This allows users to concurrently work on multiple invocations of the same application with different state variables. For example, consider a Web application which maintains information about users. Each invocation maintains a state variable representing the account ID. If dynamic argument embedding is used, a user with 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 state variables are embedded within the URL's of the application. Therefore, different invocations will have different sets of URL's which can all be cached.

By contrast, if Netscape cookies are used, it is not possible for a single browser to concurrently work on multiple invocations of the program for two reasons. Since the URL's from the different invocations would likely be the same, pages from the most recent invocation would likely overwrite pages from any earlier invocation in the browser's cache. Secondly, even if this were not the case, the cookie representing the account information from the most recent invocation would overwrite any cookie representing account information from a previous invocation.

Dynamic argument embedding provides precise control over which URL's receive state variables. State variables are only sent to hypertext links which are part of the application and which are not restricted from receiving them. By contrast, cookies can be sent to URL's which are not part of the application intended to use them. If the URL receiving the cookie is expecting a cookie of the same name, the cookie created by the unrelated application could result in a bug. Passing cookies to malicious URL's can result in security risks.

Dynamic argument embedding provides a wider range of options for ensuring that state variables are transmitted securely. A cookie has a security option which can be set to prevent it from being sent unless SSL is used. Dynamic argument embedding has a similar option which also recognizes S-HTTP as a secure method of communication. Dynamic argument embedding also has an option which automatically converts conventional HTTP requests to SSL requests in order to protect state variables. This option can be applied at the granularity of:

1. Each URL which transmits a specific state variable.

2. Each modification of a specific hypertext link to transmit any state variable (s).

Cookies may have a performance advantage over dynamic argument embedding. Our algorithm increases the load on the server mostly because of the overhead of the Common Gateway Interface. This overhead is significantly reduced with better methods of invoking server programs. Over time, API's such as FastCGI, NSAPI, and ISAPI are likely to become commonplace. As this happens, the overhead of dynamic argument embedding will diminish.

2.3 Dynamic Page Modification

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

As another example, suppose that a client is in a conversation where the names of major corporations appear frequently in the text. The client contacts a server which has access to a database of home page URL's 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. For example, each time the name IBM or International Business Machines appears in an HTML page, the server would like to convert the text string to a hypertext link to IBM's home page. By doing this, the client would be able to obtain useful information about companies appearing in the conversation by pointing and clicking. This can be accomplished by modifying the *embed1* module to search HTML pages for all company names which appear 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 component of dynamic argument embedding is that the dynamic argument embedder has access to pages before they are sent to the client. Various people have proposed proxies which sit between the client browser and the network in order to filter and analyze all incoming Web pages. Such proxies can be used to collect information about Web pages which have been viewed and detect common browsing patterns. One of the difficulties with implementing these proxies is inserting the proxy between the client and the network. Specialized proxies might be needed for different client and network configurations. In addition, anyone desiring to use such proxies must take the time to install them.

Using our technology, a server application which we will refer to as a Web site analyzer can be written which implements proxies for any standard browser connected to the World Wide Web. Users don't have to add special hardware or software to their browsers in order to use the Web site analyzer. Instead, a user would simply access the Web site analyzer Web site. The Web site analyzer would then prompt the user for a base URL representing a Web site to be analyzed. The Web site analyzer would then analyze each page accessed by hypertext links and collect relevant information before sending it to the client. Analysis would only be applied to HTML pages loaded by following hypertext links from the base URL. However, the Web site analyzer could provide options for combining information from several conversations.

3 Preserving State in a Business Transaction Server

We have implemented dynamic argument embedding for the Coyote Virtual Store which is a transaction processing system prototype developed at the IBM T. J. Watson Research Center. The Coyote Virtual Store server allows businesses to sell goods over the Internet. Customers access the Coyote Virtual Store via a standard browser. In order to communicate securely, the browser should be able to communicate using SSL. However, some services can be used by browsers which don't support SSL. Users may browse catalogs describing products and can pick and choose which items to add to their purchase lists. When the user has determined that a purchase list is complete, he commits to the purchase and is subsequently billed.

The Coyote Virtual Store maintains information about the customer, inventory, and products in a database. In addition, Coyote allows users to browse product catalogs in the middle of a session. Coyote assumes very little about the format of product catalogs. Catalogs may consist of HTML files as well as CGI programs. They may exist on local or remote servers. State information including a user ID and session ID are maintained for clients performing transactions with the Coyote Virtual Store. Dynamic argument embedding is used to maintain this state information. The architecture for the Coyote Virtual Store is shown in Figure 4.

Authentication is not required to view product catalogs. In order to purchase products, update customer information, or access certain types of information, it is necessary for the client to authenticate itself by entering a user ID and password. If the client is new to the system, the client picks a user ID, password, and provides some additional information such as an address and telephone number. Authentication is only required once per conversation, however. As soon as a user has been authenticated, the userid 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 which is passed between the client and server as a state variable. Session ID's 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 script, the server makes sure that a valid session ID was passed along with the userid to the CGI script. SSL prevents other users from obtaining the session ID and making unauthorized transactions. An alternative would be to use the client's password as a state variable for authentication. The advantage to using session ID's is that passwords are given extra protection. They are only transmitted across the network once when the initial authentication takes place. 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 client begins browsing offerings from the product catalog. At this point, communication is stateless. The goal is to defer authentication in order to entice as many people as possible to view the catalog. At some point, the client finds an item which she would like to add to her purchase

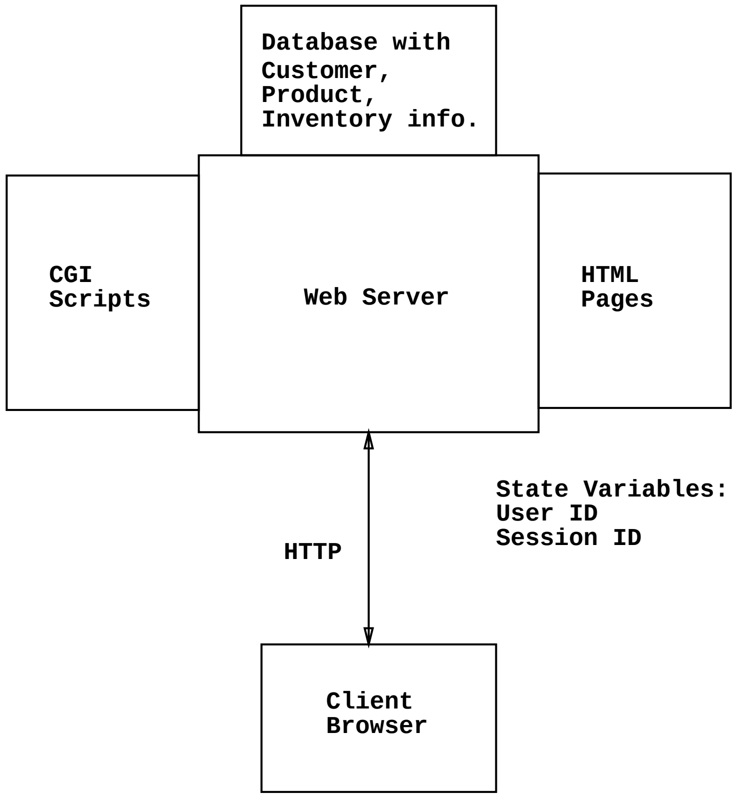


Figure 4: The Coyote Virtual Store architecture.

list. The client 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, add additional items to the purchase list, commit to purchases, or view and update database information without authenticating herself again (Figure 5).

3.1 The Advantages of Dynamic Argument Embedding

Dynamic argument embedding provides a number of advantages for maintaining state in the Coyote Virtual Store. HTML forms were not an option because conversations consist of a high percentage of pages which are not dynamically generated forms. Thus, dynamic argument embedding and cookies were the only reasonable options.

The use of dynamic argument embedding over cookies provides a number of advantages. The Coyote Virtual Store runs on any standard server which supports HTTP. In addition, any standard browser which supports HTTP can be used to communicate with Coyote. By contrast, if Netscape cookies had been used, Coyote would have to be implemented on a platform supporting cookies. In addition, only browsers which support cookies could access the features of the Coyote Virtual Store which depend on state preservation. Dynamic argument embedding allows a user with multiple accounts to concurrently access more than one account by using the browser's cache and flipping between the pages in the browser's cache. This would not be possible using cookies because the browser would not be able to maintain state information from multiple accounts concurrently.

User ID and session ID information are confidential and should not be passed to URL's which are not part of the Coyote Virtual Store. Dynamic argument embedding makes it easy to structure the application so that user and session ID's are only passed to CGI scripts which need to access or propagate them. It is harder to do this with cookies. In addition, specifying the expiration time for cookies representing user and session ID's is problematic. If cookie lifetimes are too short, they will become invalid before the application completes. If cookie lifetimes are too long, they will continue to exist after the application has completed. By contrast, dynamic argument embedding doesn't need explicit expiration times for state variables because the state variables are an integral part of the application.

4 Conclusion

This paper has presented a new method called dynamic argument embedding which preserves state on the World Wide Web by embedding state information in hypertext links of HTML pages. The algorithm can be generalized to other applications besides state preservation where it is necessary to modify or examine Web pages before they are displayed by the client's browser.

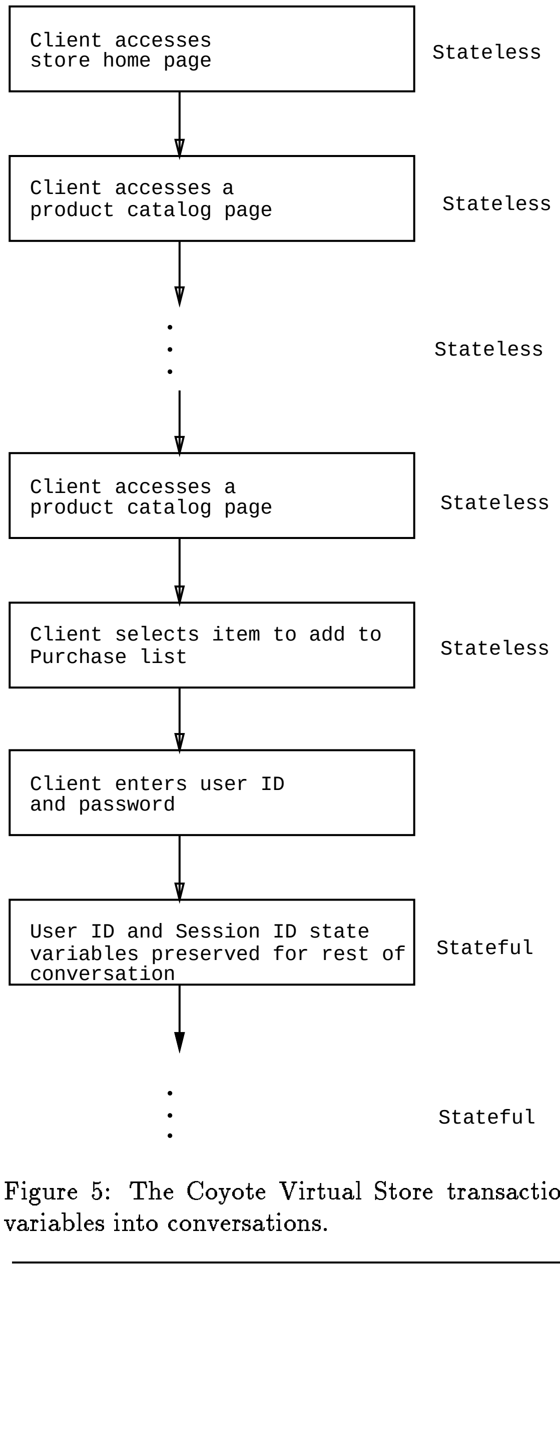

Client accesses store home page
Stateless
Client accesses a product catalog page
Stateless
Stateless
Client accesses a product catalog page
Stateless
Client selects item to add to Purchase list
Stateless
Client enters user ID and password
User ID and Session ID state variables preserved for rest of conversation
Stateful
Stateful

Figure 5: The Coyote Virtual Store transaction processing system embeds user ID and session ID state variables into conversations.

We compared dynamic argument embedding to Netscape cookies and HTML forms which are the two most commonly used methods for maintaining state on the World Wide Web. HTML forms are less general than the other two methods because every response from a server must be a dynamically generated HTML form.

Dynamic argument embedding has a number of advantages over cookies. Cookies are not part of the HTTP protocol and will only work with browsers and servers with special support. By contrast, dynamic argument embedding works all standard browsers and servers which support the HTTP protocol. Dynamic argument embedding cannot be disabled by a browser the way cookies can be. Cookies leave a persistent record of URL's which have been visited on disk and can be used by unknown parties to track Web sites that have been visited by a client. Dynamic argument embedding doesn't have either of these features.

Dynamic argument embedding provides more precise control over which URL's have access to state variables. Our algorithm has a wider variety of options than cookies for ensuring that state variables are transmitted securely. Dynamic argument embedding also allows multiple invocations of the same program with different state variables to exist concurrently by making use of client caching in situations where this cannot be done using cookies.

We described how dynamic argument embedding was used to implement the Coyote Virtual Store transaction server. The use of state variables allows users to perform multiple secure transactions while only authenticating themselves with a user ID and password once.

References

[1] K. Arnold and J. Gosling. *The Java Programming Language.* Addison-Wesley, 1996.

[2] J. December and M. Ginsburg. *HTML and CGI Unleashed.* Sams.net, Indianapolis, IN, 1995.

[3] Enterprise Integration Technologies. Secure HTTP. http://www.eit.com/creations/s-http/.

[4] I. S. Graham. *The HTML Sourcebook.* John Wiley & Sons, Inc., New York, NY, 1995.

[5] Internet Engineering Task Force. HTTP: A Protocol for Networked Information. http://www.w3.org/pub/WWW/Protocols/HTTP/HTTP2.html.

[6] A. Iyengar. Dynamic Argument Embedding: Preserving State on the World Wide Web. *IEEE Internet Computing*, 1(2), March/April 1997.

[7] Netscape Communications Corporation. Client Side State - HTTP Cookies. http://www.netscape.com/newsref/std/cookie_spec.html.

[8] Netscape Communications Corporation. The Server-Application Function and Netscape Server API. http://www.netscape.com/newsref/std/server_api.html.

[9] Netscape Communications Corporation. The SSL Protocol. http://www.netscape.com/newsref/std/SSL.html.

[10] Open Market. FastCGI. http://www.fastcgi.com/.

[11] N. J. Yeager and R. E. McGrath. *Web Server Technology.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.