Distributed Virtual Malls on the World Wide Web

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

Abstract

Virtual malls allow consumers to shop and purchase products on the World Wide Web from multiple stores. This paper presents a virtual mall in which stores may be distributed across multiple Web sites. Stores participate in the virtual mall by communicating with a mall coordinator. The virtual mall allows shoppers to perform actions across multiple stores simultaneously such as viewing product availability. Multiple purchases across different stores can be coordinated using multi-phase commits. The mall coordinator can authenticate clients on all stores participating in the virtual mall while only requiring clients to provide authentication information once. State information is preserved using dynamic argument embedding which is compatible with all browsers and servers supporting HTTP and is less obtrusive than cookies. The distributed virtual mall concept and infrastructure can be applied to other distributed electronic commerce applications on the Web.

1 Introduction

Electronic commerce is increasing exponentially, especially on the World Wide Web. Many businesses have set up virtual stores on Web sites that allow consumers to shop and purchase products on-line. Software such as IBM's Net.Commerce [4] allow businesses to establish virtual stores on the Internet. As a logical progression from these individual stores, virtual malls have also begun to appear on the Web.

As with real malls, virtual malls provide an opportunity to benefit both businesses and consumers. Businesses benefit from their visibility on the mall. For example, a virtual store which is part of a virtual mall may catch the attention of a consumer who originally comes to the mall to buy goods from another store. Consumers benefit from the opportunities for integration and coordination of goods and services on the mall. A virtual mall may provide, for example, an integrated directory of the goods and services of many virtual stores. In addition, a virtual mall may provide a single point of authentication for all of the virtual stores on the mall; a client only needs to identify itself once to perform transactions with any store in the mall. A virtual mall could also coordinate group transactions involving multiple stores.

Thus far, the virtual malls that have appeared on the Web have taken one of two approaches. First, some malls provide a Web page with hypertext links to independent stores without coordinating actions among the individual stores. It is easy to find examples of these types of virtual malls using any of the major search engines such as Yahoo! [13]. From a technical standpoint, such virtual malls are trivial to implement. Second, some malls have provided coordination features among multiple virtual stores within the same Web site. Software such as the Netscape Merchant System [8] facilitate the construction of virtual malls at a single Web site.

Each of these approaches has disadvantages. The first approach does not provide any coordination among the individual stores except for the collection of their URL's. The second approach, while providing coordination among multiple stores, is restrictive in that the virtual stores must be implemented and managed within a single Web site. This could be unacceptable to many businesses, who may want to manage their own store, but still participate in the virtual mall.

The Coyote virtual mall architecture is one in which stores may be distributed across multiple Web sites. Stores participate in the mall by communicating with an entity known as the *mall coordinator* (Figure 1). Any store on the Web can participate in the virtual mall by implementing a simple set of API functions for communicating with the mall coordinator. There are several services which the Coyote virtual mall provides:

1. Added visibility for stores participating in the virtual mall.
2. Single authentication for clients. A client only needs to authenticate itself once to perform transactions with any store in the mall. The authentication may take place while the client is communicating with a store or with the mall coordinator.
3. The ability to perform coordinated transactions among a group of stores. For example, it is possible to shop for a set of related items at multiple stores by adding items to shopping carts at the individual stores. When the client is satisfied with the ensemble of items it has selected, it can instruct the mall coordinator to commit to all purchases for all stores in the group. This obviates

having to visit each store individually in order to commit to all purchases. Multiphase commits could be used to ensure all-or-nothing semantics in purchases from several stores. For example, a client may want to purchase related sets of items from some of the stores, and may want all or none of the items; this could be achieved either by a two-phase commit process [2], or by a compensation paradigm [1].

4. The ability to aggregate information from multiple stores. For example, clients can query the availability of products across multiple stores. It is also possible to view transaction information for a client across multiple stores in a single action.

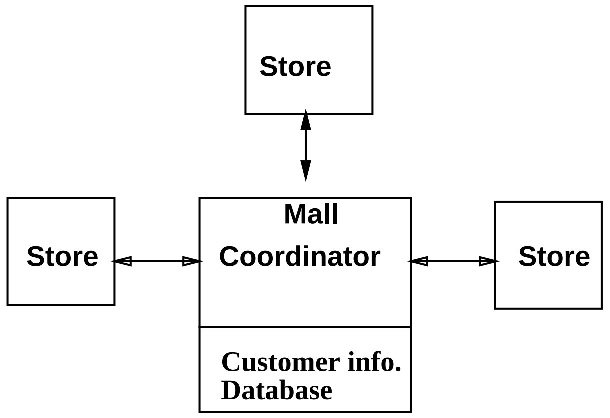

Figure 1: Stores participate in the Coyote virtual mall by communicating with the mall coordinator. Stores may be located at remote Web sites.

The remainder of the paper describes the Coyote virtual mall architecture in detail. Our findings are of relevance not just to virtual malls but also to other Web applications which are distributed, need coordinated actions among sites, or need to maintain state.

State preservation is needed by the Coyote virtual mall for authentication information as well as determining which stores have been visited during a browsing session. Cookies [7] are probably the most commonly used method of preserving state on the Web. However, cookies have a number of drawbacks. They are currently not part of the HTTP protocol and are not supported by all browsers and servers. Many clients reject cookies because they leave a permanent record of URL's which have been visited on disk and can be used by unknown parties to track Web sites which have been visited by a client. In addition, cookies usually have lifetimes that differ from the duration of the application. Because of the drawbacks of cookies, Coyote uses dynamic argument embedding [5] for state preservation, an algorithm which doesn't have these drawbacks.

2 Virtual Mall Architecture

The key feature of our virtual mall architecture is that stores may be distributed across multiple Web sites (Figure 1). Stores communicate with the mall coordinator but not with each other. Stores must implement API functions in order to communicate with the mall coordinator but do not have to know how to communicate with each other.

The mall coordinator performs three main functions. It authenticates clients so that they can perform transactions at any store by only providing authentication information once; it coordinates actions among a group of stores by communicating with the stores using the mall API functions; finally, it combines information from several sites and presents the unified information to the client.

2.1 Flow Through the System

A client may access the virtual mall either from the mall coordinator's home page or from one of the store's home pages (Figure 2). The mall coordinator presents clients with information on the various stores at the mall, search functions for items available at stores, links to shop at the stores and group functions for coordinated actions among stores. The client typically might then search to decide on which stores to shop at, and then follow links to individual stores. In shopping at the stores, clients directly connect to the stores and browse (i.e., the mall coordinator does not act as a proxy). When an unauthenticated client needs to perform an action requiring authentication at a store, such as placing an order, the store communicates with the mall coordinator in order to authenticate the client on all stores across the mall. The details of the authentication process are explained in the next section. In this manner, a client may visit several stores at the mall and build shopping carts at various stores concurrently. The client may purchase items at the stores independently, if so desired. Group functions are performed though the mall coordinator. The group functions are described in more detail later, and include functions for searching all stores, viewing shopping carts at all stores and group purchases. The group for these actions could be the stores visited by the client during this session, all the stores, or could be selected by the client. The mall coordinator sends requests to the stores in the group, merges information from the stores, and coordinates actions among them.

2.2 Authentication

Many Web applications require users to identify themselves. In order to do so, a user typically provides a user ID and password by typing them into a browser. The authentication information provided by the user can be encrypted as it is passed over the network so that the information cannot be stolen by someone snooping on the network. SSL [10] is currently the most commonly used method for transmitting secure information from a browser to a server.

An alternative to using manually entered user ID's and passwords is to use a digital certificate stored on disk or on a Smartcard. Digital certificates are arguably more secure than manually entered passwords.

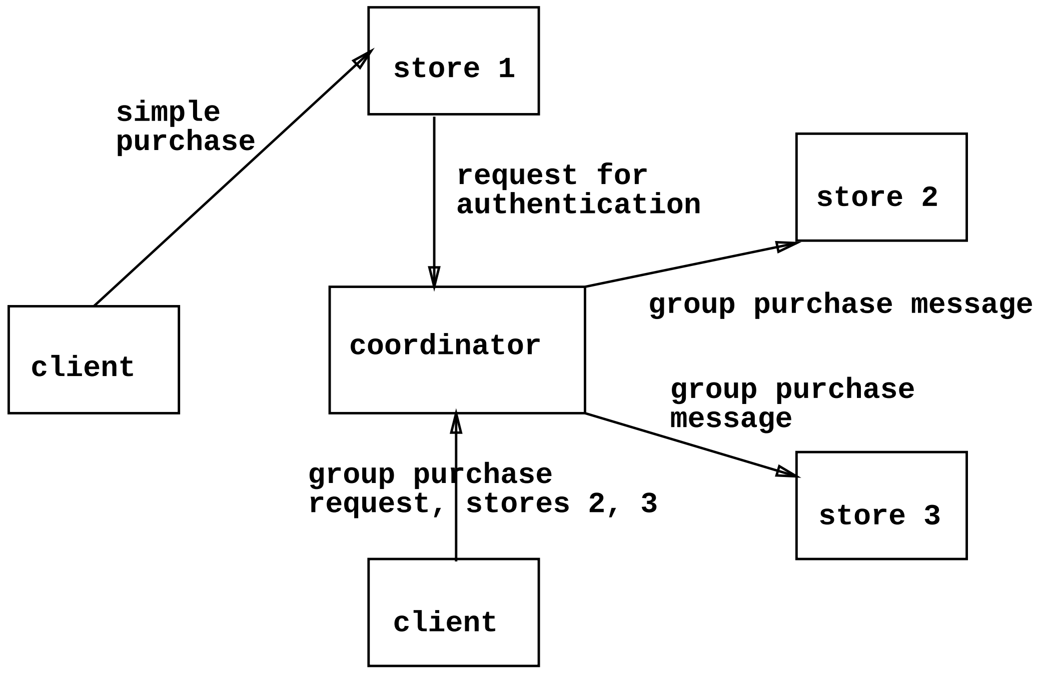

Figure 2: Communication flows through the virtual mall.

It is also possible to do things with digital certificates which cannot be done with manually entered passwords. For example, public-key cryptography can be used in conjunction with digital certificates to allow users to digitally sign electronic documents [12]. However, digital signatures require more software (and possibly hardware) support than manually entered passwords. They are not yet in common use on the Web. We chose to use manually entered passwords for authentication at the Coyote virtual mall because they are currently the common way of doing things on the Web. Many virtual stores would not be able to handle digital certificates. Digital certificates present complications at the client side as well. In the future, we may also provide support for authentication using digital signatures.

There were three main requirements for our authentication algorithm for Coyote:

1. If the client is visiting multiple stores, the client should only have to provide authentication information to the mall once in order to perform transactions with any store.

2. Individual stores as well as the coordinator should have the ability to authenticate the client. The client should not have to explicitly contact the coordinator in order to identify itself.

3. The client should not be prompted for authentication information until the information is absolutely needed. As soon as a Web site prompts a client for authentication information, the prompt inevitably turns a certain fraction of clients away. Virtual stores seek to attract as many clients as possible in order to maximize sales. When a client is merely browsing store catalog pages, the client should not have to provide authentication information. When the client tries to buy something, the virtual store must know who the client is. It is at this stage that the client should be prompted for authentication information.

One of the key aspects of the algorithm is how client state information is preserved during interactions between the client and the virtual mall. After a client provides authentication information, the virtual mall must create state which identifies the client and the fact that it has been authenticated. Otherwise, the virtual mall would have to prompt the client for authentication information every time the mall needs the identity of the client. We chose to use dynamic argument embedding for preserving state information. State preservation is discussed in more detail in Section 3.2.

The coordinator maintains a database with customer information in it. Any virtual store attempting to authenticate a client will consult the coordinator to determine if the client has entered a valid user ID and password. After the client enters a valid user ID and password, a session ID state variable is generated. Session ID's act as temporary passwords and identify the client during subsequent transactions with the virtual mall. Session ID's are passed back and forth between the client and virtual mall. For browsers which support SSL, SSL can be used to encrypt passwords and session ID's. The purpose of using session ID's instead of passwords for subsequent authentications after an initial one are to reduce the exposure of passwords. Passwords are only passed across the Web once. If a session ID is stolen, the damage which can be done is limited. The use of SSL makes it extremely difficult to steal either passwords or session ID's over the network.

Figure 3 shows what happens when a client performs an action requiring authentication at a server X. Server X may be a virtual store or the mall coordinator. Server X determines if user ID or session ID state variables exist which would indicate that the client is already authenticated. If so, X passes the state variables to the coordinator (assuming that X is not the coordinator) and the coordinator verifies if the state variables are valid. If, on the other hand, user ID and session ID state variables do not exist, X prompts the client for a valid user ID and password. Secure transmission of authentication information is provided via SSL.

After X has received a user ID and password from the client, it must verify that they are valid. It does so by contacting the coordinator (assuming that X is not the coordinator). The coordinator checks if the user ID and password are valid by consulting its customer information database. If the user ID and password are valid, then the coordinator creates and stores a session ID which is a temporary password which will be used to authenticate the client for subsequent transactions during the current session. Session ID's are chosen randomly from a large enough set of values so that they are highly unlikely to be guessed. The coordinator then passes the session ID to server X (assuming that X is not the coordinator). Server X then preserves the user ID and session ID as state variables, using dynamic argument embedding, so that they can be used to authenticate the client during subsequent transactions with the virtual mall.

The use of SSL makes it difficult for a malicious third party to steal passwords or session ID's by listening to the network. Since the session ID is passed between the client and virtual mall multiple times while the password is only passed from the client to the virtual mall once, session ID's have a greater chance of being stolen. In order to reduce the risk of session ID's being stolen, the coordinator can periodically generate new session ID's for a client making multiple transactions during a single session. In the most secure mode, a new session ID is generated after every transaction.

A security feature which we have not yet implemented is for the virtual mall to record the IP address of the client when the client first authenticates itself with a user ID a and a password. Whenever a client c implies that it was previously authenticated by providing a user ID state variable equal to a and a session ID state variable, the virtual mall would verify that the client c has the same IP address as the client originally providing the authentication information. If not, the request from c would be rejected, even if c provided a valid session ID. This security feature is not foolproof. For example, different clients can have the same IP address if they are behind the same firewall.

2.3 Group Actions

The mall coordinator coordinates actions among a group of stores. It does so by sending messages to the stores via API functions which the stores must implement to participate in the group action. After a store receives a message from the mall coordinator, the store performs the action and sends the results back to the mall coordinator (Figure 4). The process of sending messages to stores and gathering responses may be performed in multiple phases. Two-phase commits are thus possible.

Stores can reply to the coordinator with HTML text which is sent back to the client. The coordinator merges all HTML responses from stores and sends the results back to the client. The HTML text may include hypertext links and image files. The coordinator does not impose many format restrictions on store replies to group actions. Individual stores have considerable flexibility in providing output in response to group actions.

The architecture for performing group actions is quite general. The mall coordinator can support a virtually unlimited number of different group actions provided that the stores have implemented the API functions to perform the group action. Furthermore, there is no "complete" set of functions which a store has to implement in order to participate in the mall. It is possible for one store to implement group actions not implemented by another store in the same virtual mall.

Group actions supported by the mall coordinator include the following:

- Product availability. It is possible to query all stores in a group regarding whether or not they carry a certain product.
- Transaction information. A client can request information about purchases made at a group of stores in a single action.
- Group purchases. A client can shop for a set of possibly related items at a group of stores by adding items to shopping carts at the various stores before committing to purchase the contents of the shopping carts. The virtual mall allows the client to perform the following in a single action:
  - View the contents of all shopping carts in the group.
  - Empty all shopping carts in the group.
  - Commit to the purchase of all items in shopping carts. If any of the items cannot be delivered by a store, buy as many items as possible.
  - Commit to the purchase of all items in shopping carts in two phases [2]. This would mean that the coordinator would initially send a message to each store in the group asking if the store could satisfy the order for all items in the client's shopping cart. If all stores responded affirmatively, all items would be purchased. If any store responded

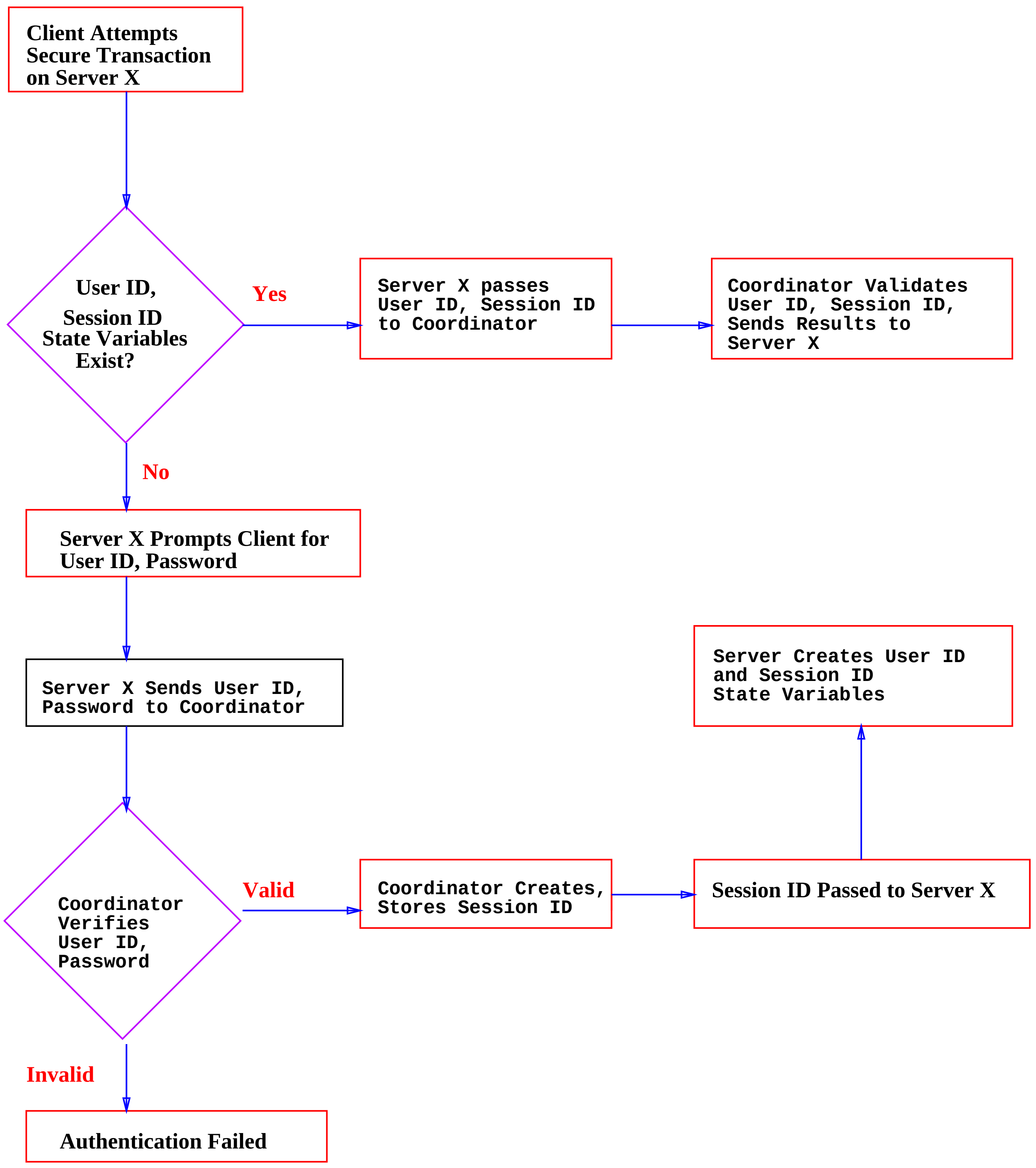
Client Attempts Secure Transaction on Server X
User ID, Session ID State Variables Exist?
Yes
Server X passes User ID, Session ID to Coordinator
Coordinator Validates User ID, Session ID, Sends Results to Server X
No
Server X Prompts Client for User ID, Password
Server X Sends User ID, Password to Coordinator
Server Creates User ID and Session ID State Variables
Coordinator Verifies User ID, Password
Valid
Coordinator Creates, Stores Session ID
Session ID Passed to Server X
Invalid
Authentication Failed

Figure 3: The authentication algorithm used by the Coyote virtual mall. A client only needs to provide user ID and password information once in order to be authenticated on all stores in the mall.

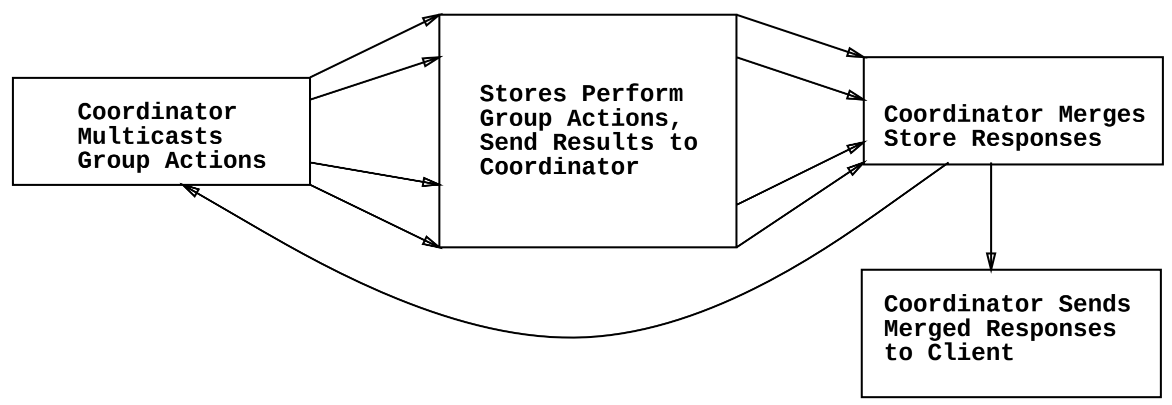
Coordinator Multicasts Group Actions
Stores Perform Group Actions, Send Results to Coordinator
Coordinator Merges Store Responses
Coordinator Sends Merged Responses to Client

Figure 4: The mall coordinator coordinates actions among a group of stores by sending messages to the stores which invoke API functions at the stores. The stores respond with messages which are gathered by the mall coordinator. The process of sending messages and gathering responses may take place in several phases.

negatively, none of the items would be purchased.

3 Implementation Details

We have implemented a prototype of the Coyote Virtual Mall in which the mall coordinator is implemented as a set of programs invoked from a Web server. It runs under AIX version 3.2.5 or higher. The coordinator uses IBM's DB2 database for storing information about customers and stores. The coordinator has a specific set of interfaces which it uses to communicate with stores. Any stores which implement the interfaces can participate in the virtual mall.

The stores and the mall coordinator communicate using HTTP. That way, special daemon processes for communication between servers are not required. The mall coordinator communicates with a virtual store by invoking a server program with appropriate arguments which is called by the store's Web server.

3.1 Virtual Mall API's

Group actions are implemented by the coordinator invoking an API function *group_action* on each store participating in the group action. This function takes at least two parameters and possibly others depending upon the nature of the group action:

- *type*1 : a string which indicates the type of action to be performed (e. g. display shopping cart contents, empty shopping cart, commit to a purchase, search for an item, etc.).

- *type*2 : an integer between 0 and 3. 0 indicates that the action is one-phase. 1, 2, and 3 correspond to two-phase actions. 1 indicates a query of whether the store can perform a transaction. 2 indicates an instruction to commit to a transaction after all stores have indicated that they can perform the transaction. 3 indicates an instruction not to perform the transaction because at least one store in the group cannot perform the transaction.

Group actions such as committing to purchases also require the coordinator to pass information identifying the client. Other group actions such as searches don't require the coordinator to pass information identifying the client.

When a store performs a transaction on behalf of a client for which user ID and session ID state information already exist, the store first verifies that the user ID and session ID are valid by invoking a function *verify_uid_sessionid* on the coordinator. If no authentication information exists, the store prompts the client for a user ID and password. The store verifies that the user ID and password provided by the user are valid by invoking a function *verify_user* on the coordinator. If the user ID and password are valid, *verify_user* creates and stores a session ID for the client.

Since dynamic argument embedding is used to preserve state, each store implements an API function *display_store_home_page* which is invoked whenever a client clicks on a hypertext link at the coordinator's Web site to the store. This function passes user ID and session ID state variables to the store if the client has already been authenticated. The function is implemented as a CGI program [6]. It could also be implemented using proprietary interfaces for invoking server programs such as NSAPI [9], ISAPI [3], ICAPI by IBM, and FastCGI [11] which are generally faster than CGI but not as standard.

Similarly, the coordinator has a CGI program *display_coordinator_home_page* which is invoked when a client follows a hypertext link on a store to the coordinator. This function passes user ID and session ID state variables to the coordinator if the client has already been authenticated.

3.2 Preserving State

The Coyote virtual mall must preserve state while customers are shopping. The state variables include the client's user ID, session ID, and stores visited during the current session. Cookies [7] are commonly used for preserving state in Web applications. However, there are a number of drawbacks to cookies. Cookies are currently not part of the HTTP protocol and are thus not supported by all browsers and servers. Another problem with cookies is that many users find them obtrusive. They are stored on disk and leave a persistent record of a visited Web site. Cookies are also used by businesses on the Web to track browsing patterns of clients. For these and various other reasons, many clients choose not to accept cookies even if their browsers support them. Another problem with cookies is that there is no way to tie the lifetime of a cookie to the lifetime of an application. The lifetime of a cookie is by default the lifetime of the browsing session. In order to override default behavior, the application creating the cookie must specify a specific date and time when the cookie should expire.

Because of the limitations of cookies, the Coyote virtual mall uses dynamic argument embedding for preserving state instead of cookies. Dynamic argument embedding doesn't have the previously mentioned limitations of cookies. Another advantage of dynamic argument embedding is that browsers can cache multiple copies of pages corresponding to the same application with different instantiations of state variables. For example, a client who has multiple accounts at a virtual mall under different user ID's can access two or more accounts concurrently by using the browser's cache and flipping between pages corresponding to the different accounts. Cookies don't allow this because the state information from the most recent invocation would overwrite state information from any previous invocation of the Web application. Iyengar [5] presents a detailed comparison of dynamic argument embedding, cookies, and a third method of state preservation, HTML forms.

Dynamic argument embedding is a general technique for maintaining state on the World Wide Web which doesn't require any extensions to the HTTP protocol. It is compatible with any client or server which supports HTTP. Dynamic argument embedding creates state variables whose lifetimes are tied to the lifetime of the application. This technique doesn't preserve a persistent record of where the client has visited the way cookies do. In addition, it cannot be disabled by the browser the way cookies can be.

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 server programs invoked by the application. It also modifies hypertext links to recursively invoke the argument embedder. A client running a Web application in which state is preserved using dynamic argument embedding will be passed hypertext links which are all calls to the argument embedder.

For example, suppose that *embed* is the argument embedder. The application wishes to preserve the state variables $x = 32$ and $y = 77$. 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/
embed?url=//www.watson.ibm.com/mail.html&
x=32&y=77">
```

When this hypertext link is selected, *embed* will modify hypertext links in *mail.html* to recursively invoke *embed* with the original URL and the state variables as arguments. A link to a server 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
/embed?url=//www.watson.ibm.com/cgi-bin/prog
&arg1=55&comma=1&x=32&y=77">
```

The string *"comma=1"* allows *embed2* to distinguish the original argument *arg*1 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*. When this hypertext link is selected, *embed* will invoke *prog* on the original argument *arg1* as well as the state variables x and y. It will then modify the hypertext links in the output of *prog* to recursively invoke *embed* with the original URL and the state variables as arguments.

The algorithm is depicted pictorially in Figure 5.

4 Conclusion

This paper has described the architecture of the Coyote virtual mall. Coyote allows stores participating in the mall to be distributed across multiple Web sites. Furthermore, the stores participating in the mall are likely to have been implemented independently without knowledge of how to communicate with each other. Stores participate in the virtual mall by implementing a set of API functions to communicate with a mall coordinator.

The virtual mall allows a client to perform transactions on any store in the mall after providing authentication information once. A client can determine product availability across several stores in one action. Clients can also commit to purchases from several stores in one action; two-phase commits can be used to provide all-or-nothing semantics for purchases among several stores.

The virtual mall preserves state using dynamic argument embedding. Unlike cookies, dynamic argument embedding is compatible with all browsers and servers which support HTTP. Dynamic argument embedding is also less obtrusive than cookies. No information is written to disk. A client browser cannot reject dynamic argument embedding state variables while it can reject cookies even if it supports them.

The concepts and infrastructure, described here in the context of the virtual mall, can be applied to other distributed electronic commerce applications on the Web. For example, instead of stores, the sites could be reservation systems, say airline, hotel and car rental, with coordinated actions such as all-or-nothing semantics across these sites.

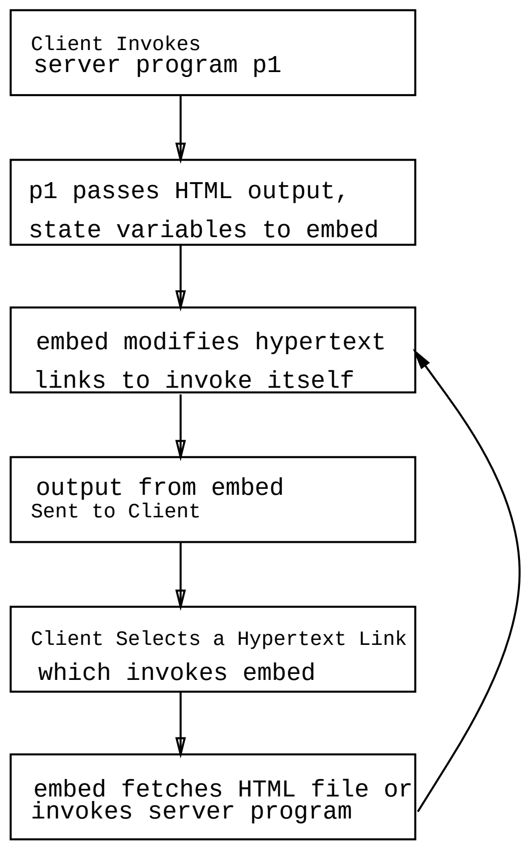

Figure 5: The major steps in dynamic argument embedding. Program *p1* is the application program which makes the decision to preserve state.

References

[1] A. Dan and F. Parr. The Coyote Approach for Network Centric Business Service Applications: Conversational Service Transactions, a Monitor, and an Application Style. In *HPTS Workshop Proceedings*, 1997.

[2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1993.

[3] C. Gross. Taking the Splash: Diving into ISAPI Programming. *Microsoft Interactive Developer*, January 1997.

[4] IBM Corporation. IBM Net.Commerce. http://www.software.ibm.com/commerce/net.commerce/.

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

[6] NCSA. The Common Gateway Interface. http://hoohoo.ncsa.uiuc.edu/cgi/.

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

[8] Netscape Communications Corporation. Netscape Merchant System. http://www.netscape.com/comprod/products/iapps/capps/mersys.html.

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

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

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

[12] B. Schneier. *Applied Cryptography.* John Wiley & Sons, Inc., New York, NY, 1996.

[13] Yahoo! Corporation. Yahoo! http://www.yahoo.com.