MultECommerce: A Distributed Architecture for Collaborative Shopping on the WWW

Stefano Puglia, Robert Carter[*] and Ravi Jain[**]
Applied Research
Telcordia Technologies
445 South St, Morristown, NJ 07960, USA
(973) 829-3178
{stef, rjain}@research.telcordia.com

ABSTRACT
The WWW has made information and services more available than ever before. Many of the first Web applications have been emulations of real world activities, in particular, e-commerce. But so far, the use of information and services on the Web has been a *solitary* one. We propose a component-based architecture for collaboration that provides *shared navigation* of the WWW along with an EJB-based server implementation. As a particular application built on this architecture, we present *MultECommerce*, through which multiple users can participate in virtual shopping trips among multiple shopping sites. MultECommerce features a multi-site shopping cart and enables one-stop checkout from all visited shopping sites. We examine security and performance issues of our architecture.

Categories and Subject Descriptors
D.2.11 [**Software Engineering**]: Software Architectures – *Domain-specific architectures, Patterns*

General Terms
Design, Performance, Experimentation, Human Factors

Keywords
E-commerce architectures, collaboration, shared navigation, e-commerce APIs, WWW engineering, component technologies, Enterprise JavaBeans.

1. INTRODUCTION
The WWW has made information and services more available than ever before. Many of the first Web applications have been emulations of real world activities, in particular, e-commerce. But so far, the use of information and services on the Web has been a *solitary* one. Every user logged in can only browse around pages and interact with them (e.g. querying databases, performing e-commerce activities, buying items on merchant sites, accessing and using various services) basically *alone*, experiencing everything on his/her[1]-own.

Instead, people like doing things together: they like to collaborate in real life, both while doing their jobs and during their leisure activities. Nevertheless they currently do not find much support for collaboration in their "on-line life".

We need to specify what we mean by *collaboration* in the work we propose in this paper. We think that collaboration on the Web is tightly bound to the concept of "getting together on the Web": to navigate around with other Web users, doing together those activities that are currently performed alone. Our basic idea is the creation of what we called *collaborative sessions* that can be joined by multiple Web users to *share* with each other *the navigation* among standard Web pages.

In our opinion collaborative activity on the Web would be ineffective without achieving this shared navigation concept. Shared navigation allows a Web user to share with other people connected to a Web based network infrastructure (Internet/ Intranet) those pages he is viewing. Such a concept could be the basis for many collaborative experiences: guided virtual museum tours in which people from different parts of the world join collective visits and follow a guide; lectures, classes, or seminars organized as a set of HTML/XML pages, where the lecturer controls the pages displayed on the audience's monitors; collective document browsing where several people browse documents together; an advanced type of on-line assistance with very complicated and rich Web sites allowing an expert to lead users and customers where they need to go.

Shopping is also something people like to do along with friends and relatives. In particular, it is likely that shopping is an activity that is *socially facilitated*, meaning that when done in the company of others people engage in it more often and enjoy it more. Considering the current boom in electronic commerce on the Web, it is interesting to try to extend the way people currently shop on the Web by adding support for more collaboration.

We present in this paper an architecture supporting a shared navigation capability and as a particular application, named

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
EC'00, October 17-20, 2000, Minneapolis, Minnesota.
Copyright 2000 ACM 1-58113-272-7/00/0010…$5.00.

[*]Author's current address: Honeywell Technology Center, Minneapolis, MN 55418. Email: rcarter@htc.honeywell.com

[**] Author for correspondence.

[1] From here on we refer to the generic user in terms of *he, him, his*.

MultECommerce, a prototype of a Web collaborative shopping system.

Our Web collaborative shopping system tries to recreate on the Web the same situation we are used to in real life: multiple people going together to different merchant sites and buying items from each of them, and optionally showing the others what they purchased. Moreover we also consider an extension of shopping on multiple sites through a "one-stop" billing process: every shopper should be able to pay for all of the items he got only once, without having to start a separate transaction with each merchant server where he bought an item.

In more general terms we transform the interaction of one user buying on a single Web merchant site into the one of many users going and buying on many Web merchant sites.

This brings with it the necessity of both support for consistent handling of the Web pages that need to be shared among collaborative participants during the navigation and the capability of interacting with an inherently heterogeneous environment where the items are purchased. Every merchant server, in fact, must remain free and independent in the definition of its data and processes.

The architecture that we present here proposes a solution to the above mentioned problems through both the definition of an adapter between multiple users and multiple sites and the suggested adoption on the merchant site of a uniform shopping interface that we have defined. At the same time our prototype, based on distributed Java components, allows Web users to avoid having to install specific software on their computers. Rather, they are able to receive everything needed directly from the Web on their browsers whenever they want to start or join a collaborative shopping session. This avoids problems with platform-specificity and software update associated with other approaches using plugins and helper applications.

2. EXAMPLE USAGE SCENARIO

Let's consider two Web users, Alice and Bob, who decide to go out shopping together. First, one of the users, say Alice, connects to a Web server providing our collaborative MultiECommerce application and initiates a collaborative session. Next, Bob, having also connected to the server, sees that Alice has created a session and decides to join Alice in that session. Bob then chooses to "follow" Alice: this means that the pages that Alice visits will be displayed in a frame of Bob's browser. Alice, in turn, decides to "follow" Bob and they then both agree on going out shopping together using MultECommerce features. Now, suppose Alice decides to go to a Web merchant server "WebCD" to purchase some audio compact discs. Alice registers on "WebCD" and buys two CDs (Pat Metheny's "Letter from home" and Incognito's "Tribes, vibes and scribes"). In the meantime Bob has gone to the Web merchant site "Joe's Bookshop" and has selected two items there ("Les Miserables" by V. Hugo and "Brave new world" by A. Huxley). Since Alice is "following" Bob, Alice can see where Bob is and, wanting to buy a book, joins Bob on "Joe's Bookshop" site and buys a book (Bruce Chatwin's "What am I doing here?"). Up to this point, Alice cannot see what Bob chose so far, nor can Bob see what Alice selected; they can only follow each other on their respective surfing around the Web. At this time, Alice decides that she is interested in letting Bob know the two CDs that she chose on "WebCD". So, she marks both her CDs *public*, meaning that they immediately become visible to Bob. Seeing "Letter from home" on Alice's shopping cart, Bob, who is a great Metheny fan and remembers he does not have that CD in his collection, decides to go to "WebCD" and get one for himself too. Bob also decides to mark the item public. While Bob goes to "WebCD", Alice decides she wants to finish her shopping: she navigates to site "Clothes-R-US" and buys a new pair of shoes.

So far this is the situation:

- **Alice** has Pat Metheny's "Letter from home" CD (*public*, Bob can see it), Incognito's "Tribes, vibes and scribes" CD (*public*, Bob can see it), Bruce Chatwin's "What am I doing here?" book (*private*, Bob cannot see it) and a pair of shoes (*private*, Bob cannot see it);
- **Bob** has Pat Metheny's "Letter from home" CD (*public*, Alice can see it), V. Hugo's "Les Miserables" book (*private*, Alice cannot see it), and A. Huxley's "Brave new world" book (*private*, Alice cannot see it).

Both Alice and Bob decide it is time to check out and pay. Each of them clicks a special button and the billing is performed in a "one stop" process, thereby avoiding their having to carry out separate transactions with every merchant they have visited (three transactions for Alice on "WebCD", "Joe's Bookshop", "Clothes-R-US" and two transactions for Bob on "WebCD", "Joe's Bookshop").

3. SYSTEM ARCHITECTURE

3.1 Requirements

The depicted scenario introduces new questions and problems to deal with. First, we have the need of navigating around together. Alice sees where Bob is going and vice versa as the two of them "share" navigation on the Web. We then have the sharing of part of the purchased items (i.e. the *public* ones) no matter where they were bought. We define *public* items in a collaborative shopping session to be all of the items the user purchased from any merchant, that are *visible* to every other user in the collaborative session. All of the other items are considered *private*.

We finally want both Alice and Bob to be able to pay for all of the chosen items just once, not having to repeat the payment n times (once for each of the n visited merchant sites where they purchased at least one item) as they currently do in the usual e-commerce.

3.2 Shopping Mediator and Multicart for Public and Private Resources

To provide a collaborative shopping session we have chosen a client-server architecture (see Figure 1) with an intermediate server playing the role of a mediator. The *Shopping Mediator* is an adapter, which works between an arbitrary number of Web users and multiple merchant servers.

We considered adding WWW-proxy functionality to the Shopping Mediator and then funneling all the HTTP traffic through a proxy on the Shopping Mediator. However, because this introduces additional load on the server, which may result in a performance bottleneck, and because the central server is a single point of failure, we chose to let the clients directly interact with the back-end server through HTTP while browsing HTML pages. We use RMI to let the clients interact with the Shopping

Mediator and to allow the latter access to the various merchant sites (see Figure 1).

The main role of the Shopping Mediator is to mediate requests from users, allowing their interaction with the heterogeneous environment represented by the merchant sites (i.e. WWW servers). The Shopping Mediator also maintains important information about the active collaborative shopping sessions and the state for each client.

The state information maintained for every user is the following:

- the location the user is visiting, expressed in terms of the URLs of the page (or pages in the case of frames) he currently views in his browser;
- all of the items the user buys on the merchant sites he visits.

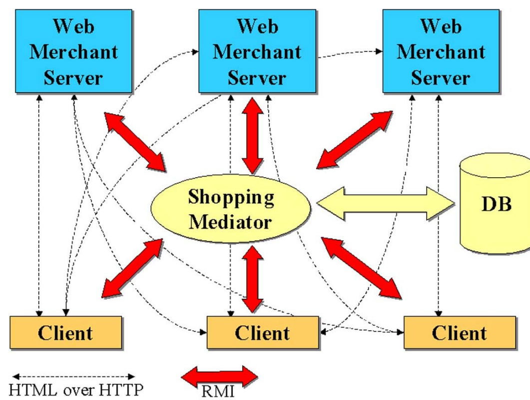

Figure 1. An overall view of the MultECommerce architecture

The first item is required to implement shared navigation. We keep track of the URLs that are downloaded by the users, so that everybody else in the same session can use them to visit the same pages on their browsers.

As for the sharing of the purchase, every client of the system may collect items from the different merchant sites he visits. These items are maintained as usual, on the various sites, in a virtual shopping cart the client has on that site. And, as usual, every site provides its own way of implementing this virtual cart and is totally independent in the handling of the items maintained on it.

The collection of all of the virtual carts the user has on every site he visits during a collaborative shopping session represents a new concept we introduced as a solution to the problem of sharing purchased items with others. We refer to this multi-site shopping cart as the user's *multicart*. This multicart contains every item the user selected, no matter where from and is physically kept up-to-date on our Shopping Mediator.

In traditional e-commerce every shopper can access only his own virtual cart on a merchant site. Moreover, WWW merchant sites do not support public and private resources for a user. Everything is automatically private since there is no concept of sharing.

Duplicating on the Shopping Mediator all of the items a user purchased during a collaborative shopping session, although redundant, has the advantage of freeing the access to the purchase from the merchant sites' rules. In fact, once the items are recorded on the Shopping Mediator, users can access them directly following rules that we (and not the merchant sites) implement. In particular, as the example shows, we can decide to implement the "typing" of items: every user can mark *public* all of those items he wants to share with others in the session. Similarly he can decide to keep *private* those items he does not want to share.

3.3 Shopping Cart and Merchant Server API

Since users shop directly on the merchant servers, the Shopping Mediator is not notified when a user has added an item to his cart on a merchant server. The Shopping Mediator must be able to efficiently update its multicart view by querying all of the merchant servers that the user has visited. This update happens whenever a user makes a request to view his multicart. As a matter of fact, the update happens both when the shopper himself asks to view his own cart and when someone else wants to see the public part of a shopper's multicart.

To facilitate the multicart update, we must assume a uniform interface to the shopping carts on the back end merchant servers. Further, we have to provide the capability to handle the interaction with an inherently heterogeneous environment. Every time the Shopping Mediator makes a request on a merchant server, on behalf of a collaborative user, it must know how the user is identified on that merchant server. We assume there is a unique way to identify every shopper on a merchant site. It is common practice on the WWW today that every user, who wants to buy on a specific merchant site, registers there in order to be identified during his visit.

Every time a collaborative shopper receives an "id" on a merchant site, he passes the "id" for that site to the Shopping Mediator. These "ids" serve as pointers to the shopping carts, which are maintained on the merchant servers. The Shopping Mediator can then use them to access the individual carts on the merchant servers.

In order for the Shopping Mediator to access in a consistent way the variety of different merchants, we propose an interface to be adopted by all of the merchants wanting to participate in our collaborative shopping activity and a well defined "object" which is exchanged through this interface. We defined a *ShoppingCart* object and a *Merchant Server API* and implemented a first version of them in Java, as explained below.

While there may be some resistance on the part of merchants to adopting a common API, we believe that the same reason that brings competitors together in the physical world will bring them together in the virtual world: that is where the customers are. Many vendors of similar items compete successfully in shopping malls all over the world and their presence there exposes them to many more potential customers. In addition, common APIs are already being developed for B2B e-commerce [1, 9] and while ours may not be the one chosen as the B2C API, certainly one will emerge.

The API is defined by three basic methods: one method retrieves the current state of the cart on a server and is used by the Shopping Mediator to keep its multicarts up-to-date; another method resets the cart on the merchant server and is used when the user desires to delete an item from the multicart; finally, a third method represents a uniform way of purchasing all the items in the cart on a merchant server. For purchase, we assume that all the necessary credentials, authorization, etc. can be encapsulated in a specific parameter.

It is important to understand the process behind this "buying" method. Essentially we are defining a distributed transaction on an arbitrarily large set of heterogeneous merchant servers. The desired "one stop billing" is actually that process that causes the Shopping Mediator to call, on behalf of the individual collaborative shopper, the "buying" method on every merchant's API to start the local process of payment there.

3.4 The Client Side

The client accesses the system using a normal Web browser. When he wants to join a collaborative shopping session, he downloads a multi-framed page providing the MultECommerce service (see Figure 2). The basic layout of this multiframed page is the following:

- a frame at the top of the browser (Applet Frame) containing a Java applet and a JavaScript representing the panel through which the user registers, joins collaborative sessions, interacts with the others, follows the others on the Web, views the others' "public" shopping cart, updates his own multicart, checks out and quits the collaborative sessions;
- a frame in the middle (Navigation Frame) where the user can navigate around the various Web merchant sites and possibly view or manipulate his site-specific shopping cart as he currently does according to the merchant site's rules;
- several subsidiary windows (or frames) that are "over the shoulder" views of what other users in a session are viewing. This provides the shared navigation capability.

The core of the client side is the top frame that contains all the machinery needed to perform this collaborative activity consistently. It is worth noticing that the user does not need to have anything else but a Java-enabled browser on his machine.

4. SYSTEM DESIGN AND IMPLEMENTATION

4.1 Shopping Mediator

We decided to use an Enterprise Java Beans (EJB) component-based approach to develop and implement the functionality of our intermediate server. EJBs allow the development of highly portable server-side business logic while EJB servers typically provide Web servers and other useful facilities.

Our Shopping Mediator can be seen as having three separate layers (see Figure 3). The application layer, that implements the "business logic" of our application and is the core of the server, is represented by a stateful session bean, called a *SessionAgent* bean. This works as an agent, performing everything on behalf of the client it is tightly bound to for the duration of the collaborative session. This is possible thanks to the ability of stateful EJBs to maintain conversational state. This means that the instance variables of the bean class can cache data, relative to the client, across method invocations. We exploit this property to store, in an "ad hoc" structure, pairs of the form (*MerchantURL, UserID*) representing the user's identifier on a merchant site. The SessionAgent bean is conversational, but not persistent and interacts directly with the data layer, which can be seen as a component view of the underlying relational database that represents the actual persistence layer. Three entity beans, persistent by definition, make up the data layer implementing the "business objects" of our application: *Resource, User* and *CollabSession*. Currently all of these beans are deployed and run on an EJB-enabled application server (BEA WebLogic 4.0.3). Further work will include porting and testing on other EJB platforms.

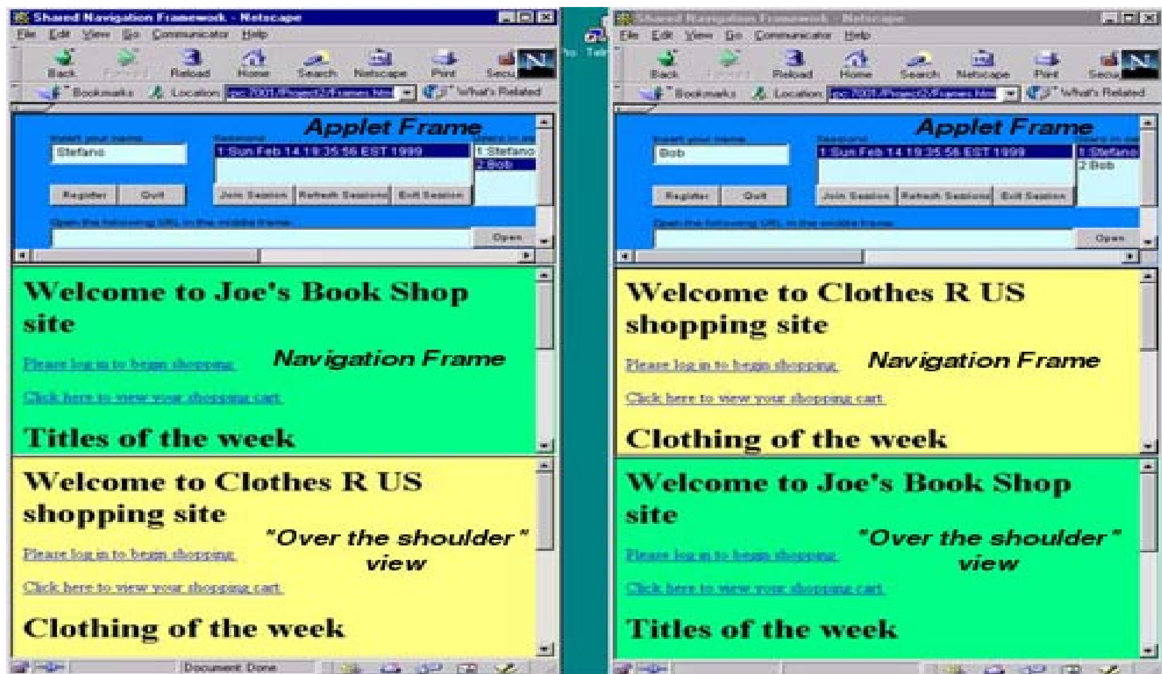

Figure 2. A snapshot of our prototype with two browsers opened

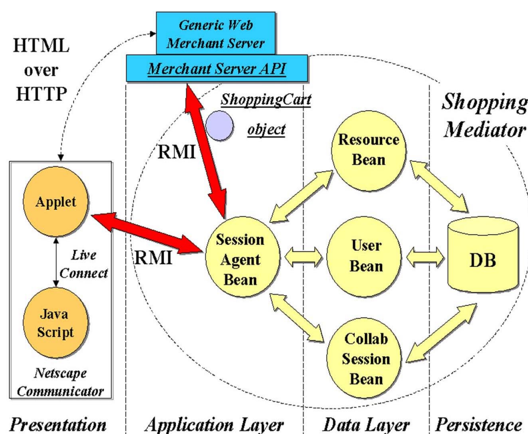

Figure 3. Architectural components

4.2 Client

As we said in the previous section, the client side of our distributed application is currently represented by a Java applet and various JavaScripts (see Figure 3). The applet represents the bridge between the client and the Shopping Mediator application layer and interacts directly with the Session Agent bean. In order to access and extract from the visited Web pages all of the information needed to allow shared navigation, we took advantage of the Live Connect/JavaScript interaction that Netscape Communicator (but not Internet Explorer) provides. For the moment our only assumption on the client side is the use of a Netscape Communicator 4.x browser.

The basic idea that we implemented is the following: a previously authorized and certified Java script sitting in the top frame "snoops" on where the collaborative user is surfing in the middle frame and retrieves the URLs of the downloaded pages. It then passes them to the applet that updates, in turn, the Shopping Mediator database through the component architecture. Similarly, the applet is able to receive the various URLs of all the collaborative users and to display them in the appropriate "over the shoulder" views to indicate the other session members' navigation. This is accomplished through a JavaScript script generated "on the fly" thanks to Netscape LiveConnect. The same mechanism is used to pass the (*MerchantURL, UserID*) pairs to the Session Agent.

4.3 Security and Privacy Issues

In our current implementation of the architecture every client pulls the information out of the server, following a pure asynchronous distributed client-server paradigm. This, though less scalable, avoids problems with security matters. In fact, we already designed a "push" method, achieving a "peer-to-peer" distributed architecture (based on RMI) in which the server (i.e. SessionAgent on the Shopping Mediator) refreshes and synchronously updates the necessary data on all of the participating clients. This is done through the interaction of remote objects both on the client and the server side. The problem we encountered implementing this in Java is due to restrictions imposed by the Java security model on remote access to an applet. Firstly, the absence of support for multiple inheritance in Java made it impossible to have an applet, which already inherits from the Java Applet class, also extend the Unicast Remote Object (which is necessary to make an object remote). Secondly, there seem to be severe security obstacles preventing the server side from opening a separate socket to actively "write" on a remote object bound to the applet. The problem is that an applet may not open a server socket to listen for requests from arbitrary hosts.

In developing the client side, efforts were also made to relax certain other limitations imposed by the Java security sandbox on some needed operations. In particular, we refer here to the "snooping" activity of a JavaScript in the top frame. JavaScripts running in a frame can, by definition, access the properties of documents downloaded in the other frames only if these documents come from the same domain the script comes from. We found that only certified scripts could obtain further privileges such as allowing the "snooping" of pages coming from different domains. Without a certification process during which the user grants certain requested privileges to the downloaded components, the shared navigation, which can be seen as a sort of authorized spying, would not have been possible. For our prototype we appropriately enable the Netscape browser to permit this in order to avoid using certificates during development.

As to privacy, we realize that our mediated design offers opportunities to enhance or degrade users' privacy. On the one hand, the Shopping Mediator enables the possibility to provide anonymity to shoppers since it could prevent cookies and other private data from being transmitted to the merchants, acting as a filter between the two. On the other hand, the information that is available to the Shopping Mediator relating to user identity, preferences and habits would be of great value to merchants and marketers. Though not specifically addressed by our prototype, our inclination is that anonymity be provided to users.

4.4 Back-end Merchant Web Servers

Our system architecture includes Merchant Servers, which are the sites that shoppers will interact with using the system. We assume that the Merchant Servers present a standard HTML interface served over HTTP, hence, normal browsing is carried out directly between the clients and the Merchant Servers, as is the case in typical e-commerce systems.

Both Figures 1 and 3 show the two communication paths defined by the system for communication among clients, the Shopping Mediator and the Merchant Servers. First, there is the standard HTML over HTTP browsing activity. Second, there is the channel between the Shopping Mediator and the Merchant Server, which we have implemented over RMI.

Providing a shopping multicart and one-stop billing require that the Shopping Mediator be able to retrieve the shopping carts from the Merchant Servers on behalf of each shopper and to perform transactions with them.

For interoperability, a standard definition for a shopping cart is required. We have chosen to define a class (ShoppingCart) as a composition of Items (see Figure 4). This assumption simplifies implementation in Java, because every instance of the ShoppingCart class can be communicated using RMI (see 7 in Figure 3).

Given a ShoppingCart object, what remains is the definition of an API that the Merchant Server will export in order to allow access to the shopping carts by the Shopping Mediator. In the prototype, we use a Java interface to define the Merchant Server API.

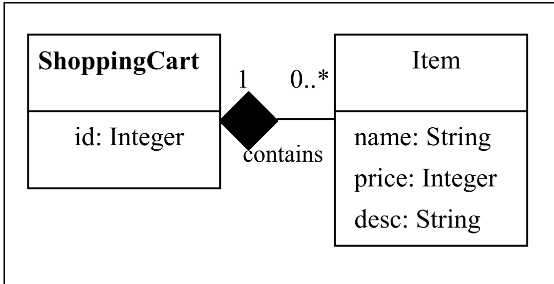

Figure 4. *ShoppingCart* and *Item* classes in the UML notation

These are the three methods currently implemented in the API:

1. **getShoppingCart** is used by the Shopping Mediator to retrieve a user's ShoppingCart from a Merchant Server; calling this method on every visited merchant site, using the user's id on that site, allows the Shopping Mediator to retrieve the shopper's multicart;
2. **putShoppingCart** resets the cart of an identified user on a Merchant Server and is used when the user desires to delete items from the multicart;
3. **buyShoppingCart** is a uniform way of purchasing, on behalf of a shopper, all the items in the cart on a Merchant Server, using the appropriate payment information.

5. PERFORMANCE RESULTS

In this section we report on performance measurements of our prototype. We focus on two issues: the overhead of our central-server design and system scalability.

Measurements were made on an instrumented version of the MultECommerce application. Recall from Section 4 that the Shopping Mediator consists of presentation, application[2], data and persistence layers (see Figure 3). Timestamps were recorded before and after method calls at the interfaces between system layers. This allows us to measure separately the performance of each layer of the architecture. Starting from the top, we instrumented the applet at the presentation layer, then the adapter and SessionAgent at the application layer. We were unable to instrument the lowest layers of the system since they are part of the EJB server.

The empirical results reported in this section are based on data collected on our local network. The Shopping Mediator was run on a desktop PC (with a Pentium II at 400 MHz and 128MB RAM aboard), while two Merchant Servers were run on dedicated laptops (Compaq Armada 7800 with Pentium II at 366MHz and 96 MB RAM), all three of these connected to a common Ethernet hub. Client machines were located on the same LAN as the mediator and servers.

For obvious reasons, we could not drive the MultECommerce applet/browser manually. We therefore implemented a version of the applet that executed a scripted series of operations against the Shopping Mediator to ensure a consistent and repeatable offered load for our performance measurements. The script performs multiple iterations of Register, Join Session, Exit Session or getShoppingCart commands as appropriate for each experiment. While measuring shopping cart retrieval performance, the script first adds a specified number of items to the shopping cart (from 2 to 100 items as shown in the performance graphs below). In this way we simulated multiple simultaneous users of the MultECommerce server.

First, we report performance results for the operations of Registration, JoinSession and ExitSession. Next, we examine the effect of shopping cart size on the response time of the getShoppingCart operation.

The basic operations on the MultECommerce application are Registration, JoinSession and ExitSession. We present measurements for 100 iterations of each operation. Each graph displays three curves for total delay at the presentation (Applet) and application (Adapter and SessionAgent) levels plotted against the iteration index.

The Applet delay curve indicates the response time seen by a user of the system and it includes the Adapter delay on the Shopping Mediator.

The Adapter delay curve represents the delay due to the Shopping Mediator and includes the SessionAgent delay. The difference between the Applet delay and the Adapter delay for a given operation measures applet processing plus network delay between the client and the Shopping Mediator.

The SessionAgent delay curve represents the delay due to the Shopping Mediator's data layer and persistence layer. For a given operation, the difference between the Adapter delay and the SessionAgent delay measures the overhead due to our use of the Adapter. In our experiments this was found to be uniformly a small value of 10-20 milliseconds for Registration. Thus the decision to use an Adapter object does not introduce excessive overhead and substantially reduces applet download time.

5.1 Registration

In order to use the MultECommerce application a user must first register with the application. The user provides a name and receives a list of existing sessions. The Shopping Mediator creates an entry for the new user and returns the list of sessions.

The delay due to registration is shown in Figure 5 for each of the three layers. First, note the startup cost associated with the first invocation. This is due to the EJB server activation and is common to all the graphs we present. In this experiment, there was only a single session returned to the client. Except for some variation in the Applet measurements due to variable network delay the overhead is about 100 msec.

In Figure 6, each new registration is followed by the creation of a session so the number of sessions increases linearly with the

[2] In fact, the application layer itself consists of two sub-layers. A thin adapter layer (which we refer to from here on as the Adapter layer) was needed "before" the SessionAgentBean to allow the applet to communicate more easily with the BEA WebLogic server. The alternative required downloading 6 MB of class files with the applet, which would be unacceptable.

number of registrations. In this case, we find that the delay at the Applet increases with the increasing number of sessions, while the delay on the Shopping Mediator remains the same (about 100 msec). The increased delay at the Applet is due to the need to render the list of available sessions in the applet GUI.

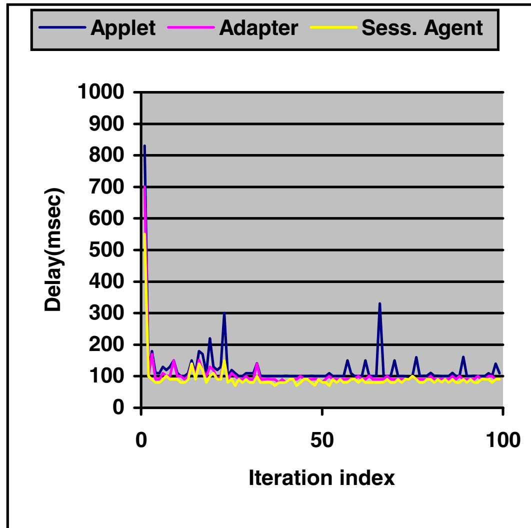

Figure 5. Registration delays (single session)

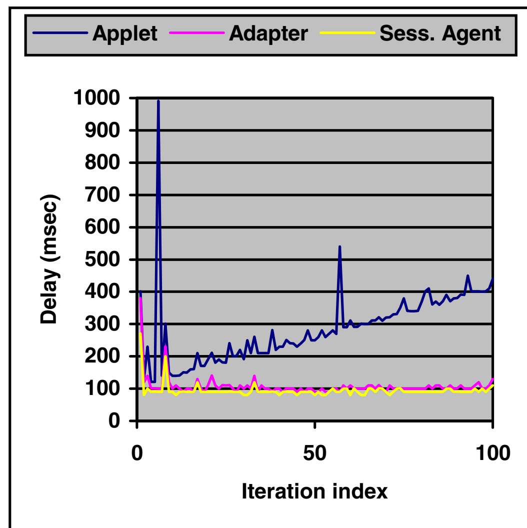

Figure 6. Registration delays (multiple sessions)

5.2 Join Session

Upon registration, a client is presented with a list of available sessions. The client may choose to join one of the existing sessions or create a new session. Figure 7 plots the delay at the three levels we measured for the case when there are no other sessions and a user creates a new session. In this case, we see that the measured difference between the Adapter and Session Agent is larger than in the case of Registration. This is due to the casting of the retrieved session object to a SessionAgent remote interface (an extension of the EJB Remote interface). When the number of sessions increases, results observed were similar to those in Figure 6: the Applet delay increases linearly with the number of established sessions due to increased rendering time.

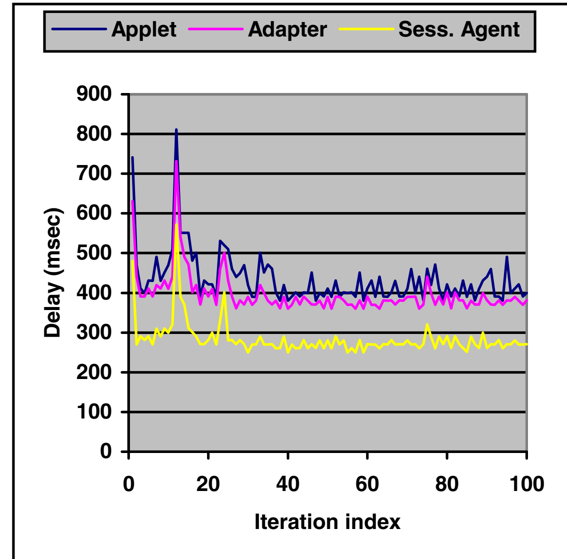

Figure 7. JoinSession delays (single session)

5.3 Exit Session

When users are ready to leave a session they invoke the ExitSession operation. Delays for the three architecture layers for this operation are shown in Figure 8. Again, we show only data for the single session case because the increase with number of sessions follows a similar pattern to that seen for Registration (Figure 6).

5.4 getShoppingCart

Managing the users' multicart is the other focus of the MultECommerce application. The application provides the getShoppingCart operation with which a user can view his multicart. The getShoppingCart operation iterates through the list of (*MerchantURL*, *UserID*) pairs and for each one retrieves the user's shopping cart from the Merchant Server and installs the contents into the user's multicart on the Shopping Mediator. The multicart is then returned to the calling client for display by the applet. In this context we study performance as the size of the multicart increases. In our test environment with two Merchant Servers we performed experiments with shopping carts ranging from 2 items up to 100 items. For our experiments we split the items evenly between the two Merchant Servers, half from each one.

In Figure 9, the total delay for the getShoppingCart operation is plotted for increasing values of shopping cart size. Each bar depicts the contribution to the total delay from the SessionAgent, the Adapter and the Applet. For each shopping cart size, the bar represents the average over 100 trials. The data in Figure 9 indicate an increasing delay as the shopping cart size

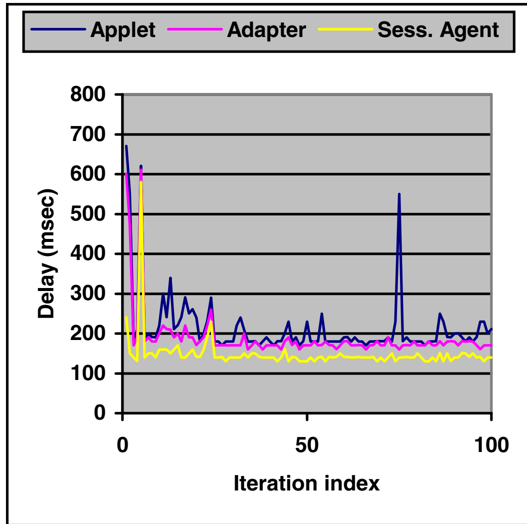

Figure 8. ExitSession delays (single session)

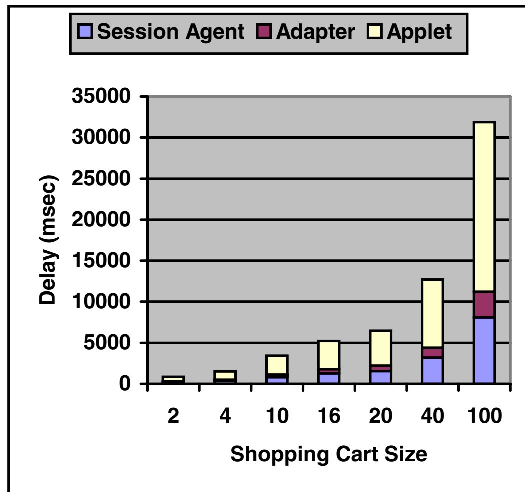

Figure 9. getShoppingCart absolute delay (by layer)

increases. This is expected, as more data must be processed with a larger shopping cart. More precisely, we expect that the time should increase at least linearly with cart size. In Figure 10 we plot the relative contribution to the total from each layer. This indicates that the size of the shopping cart has no effect on the proportion of time spent at each level of the system, supporting our hypothesis of linear scaling with cart size. To test this hypothesis, we performed a least-squares regression of a simple linear model where the total delay depends on the shopping cart size. We used the results for cart sizes from 2 to 40 as input to the regression. We observed a very good model fit with a coefficient of determination of 0.98. In Figure 11, we plot the measured data points and the computed regression line. Taken together we find that the scalability as the shopping cart size increases is just as expected.

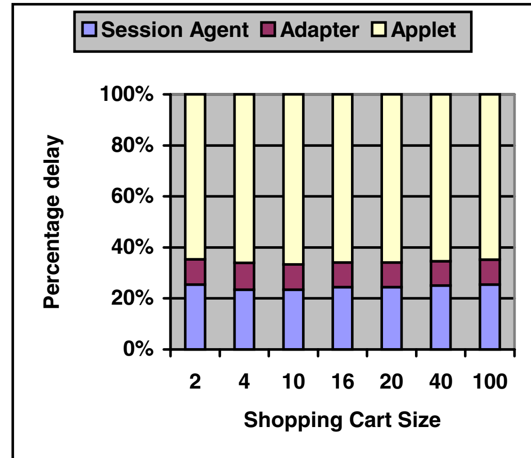

Figure 10. getShoppingCart percentage delay (by layer)

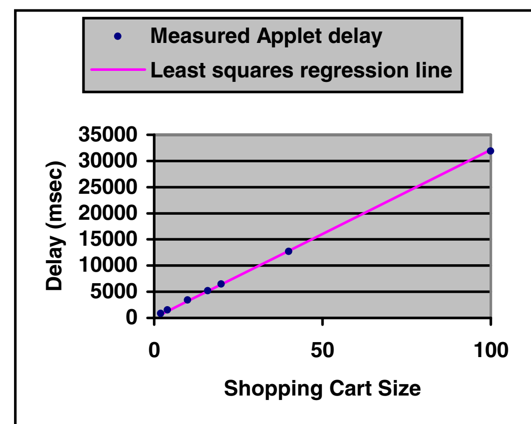

Figure 11. Applet delay Least-Squares Fit and data points

Further analysis of the getShoppingCart method reveals that the bulk of the time is spent building the shopping cart beans on the Shopping Mediator. Resolving the Merchant Server names and URLs and retrieving the shopping carts are comparatively lightweight operations that we found to be insensitive to cart size.

6. RELATED WORK

There has been a tremendous amount of previous work on creating collaborative work and interaction spaces across networks, with notable contributions from the fields of Computer Supported Collaborative Work, Groupware, and Computer Human Interaction. In particular there has been a substantial amount of work in creating multimedia collaboration spaces (offering, e.g., network conferencing and discovery of other users and ongoing collaboration sessions), application sharing, and e-commerce. It is not possible within the space of this article to fully contextualize our work within this very large background. However, we can state that in general our work primarily differs from previous work in that: it concentrates exclusively on the WWW and it allows users to stay within a single collaborative space while surfing and performing transactions across multiple

independent Web sites. In addition, secondarily, our work offers an approach and combination of features that, to our knowledge, have not been explored previously. These include: a split browser allowing each user instantaneous views of other users' frames while allowing free navigation within his own; use of an applet and Javascript so that no special client software has to be installed by the user for this purpose; a single check-out transaction across multiple Web sites; and shared spaces, portions of which can optionally be marked private.

In this section we briefly mention the previous work that, to our knowledge, comes closest to the work presented in this paper.

The retailer Land's End [6] recently introduced their "Shop with a Friend™" service, which provides a limited amount of shared navigation capability. To quote from their web page: "This service allows 2 shoppers to browse together and add items to a single shopping bag". Exactly two users can log into the Land's End web site and download an applet that invokes the shared navigation between their two browsers. Unlike our approach there is only one navigation window displayed on each browser, and these two windows are kept synchronized by the server. The two users share one shopping cart and each may add items to it. In contrast, our system supports multiple users in a shopping session each of who may travel to multiple web merchants and provides a multicart in which items from the multiple sites can be saved and later purchased with one action. The appearance of this service at Land's End encourages us that such collaborative activities have a promising future.

Several industry consortia are also developing standards for cyber-transactions. For instance, the Open Buying on the Internet Consortium (OBI) is defining standards for business-to-business Internet commerce. The OBI specification (2.0) [9] specifies the use of HTML over HTTP (using SSL v3), with Public Key certificates (X.509). Objects are currently defined using X12 EDI syntax although migration to XML is contemplated. Another effort is the Common Business Library [1] from CommerceOne that uses XML to define purchase orders and their elements (address, price, quantity, etc.). XML is an attractive technology since it allows for the use of self-defining business documents.

XML would be a more flexible choice for a commercial implementation of our system as well, allowing us to take advantage of efforts to define XML objects for e-commerce. Having an XML version of the shopping cart and the Merchant Server API implemented as something accessible directly to HTTP (i.e. servlets) would make our architecture more portable and in line with more traditional Web technologies.

Other Java-based collaboration frameworks have also been developed in recent years. Some of them in particular represent the attempt to exploit through applets the *network-centric computing* to achieve forms of collaboration not necessarily e-commerce oriented.

Both Promondia and JETS [5,11] focus on the management of real-time collaborative sessions and the definition of roles within sessions.

Promondia [5] provides a client-server architecture, based on the distribution of applets, to support collaborative tasks such as text-based chat, shared whiteboards, voting and surveys and implements a system where the interaction is limited to one Promondia server distributing pages and services.

JETS [11] gives also interesting session management policies, with a chairperson extracting and modifying specific information about participants and interesting performance results of Java applets handling images (jpegs and gifs) and video streams (H.263) during multi-participants' collaborative sessions.

Pavilion [7], an object-oriented middleware framework for developing collaborative-based applications, is a rather complete framework that let developers support features like our shared navigation in a reliable way through a well-defined Multicast IP Protocol. Moreover, even if sessions do not seem to be clearly and explicitly defined, a Leadership Protocol allows a reliable definition of users' roles. Although using applets distributed in a Web environment, some Pavilion-based applications seem to need other "worker threads" in the form of stand-alone Java application, therefore not being fully in the spirit of network-centricity.

The exploitation of the Java "mobile" technology to perform several collaborative activities seems to be the big common ground that we share with these cited works. Nevertheless the needs of interacting with a rather complex set of heterogeneous environments to achieve our multi-shopping goals shifted us more towards the investigation of architectural aspects and prompted us to propose a way to integrate the diversities we have to deal with. The other works, on the other hand, though "simple" from an architectural point of view, pay more attention to general collaborative activities and propose ways to perform them in a reliable and effective manner.

7. CONCLUSIONS AND FURTHER WORK

We have presented MultECommerce, a prototype aimed at supporting an extended form of electronic commerce on the Web. We are capable of simulating a sort of collaborative shopping on merchant sites as we believe that increased support for collaboration and cooperation are logical next steps in the evolution of the WWW. Integrating different applications is a challenging and difficult task. In our case we decided to propose to on-line merchants the adoption of a standard interface to make possible a uniform handling of items bought from different places. The main role of our Shopping Mediator is to use this interface to support multiple users in shopping together on-line much as they do in real life. We found the overhead of the central-server architecture to be acceptable for typical use while the system scales linearly with increasing shopping cart size.

Our current prototype relies on clients using the Netscape browser. This is a limitation of the prototype but not one in principle, since similar software could be created for the other popular browsers.

It is interesting to sketch out possible further uses for the Shopping Mediator. As a server side application we might want to be able to use it as a sort of a virtual Web store for different kind of information about customers. One idea could be to keep track of everything a registered buyer bought and then suggest related items. A user could also use such a "virtual closet" to co-ordinate clothing items.

In addition, considering that XML has become an essential component also in the Java world as an independent means for enabling business-to-business information interchange among heterogeneous parties, we are working to provide a new version

of our architecture, which uses XML-DTD over HTTP as its unique and uniform way of exchanging data.

ACKNOWLEDGMENT

The authors gratefully acknowledge the contributions of Paolo Missier, Telcordia Technologies, to this project through useful discussions and suggestions while conceiving the idea and during the system design and for his reviews and comments of the paper drafts.

REFERENCES

[1] Common Business Library (CBL) Version 2.0, CommerceOne, http://www.commerceone.com, 1999.

[2] Ed Roman, *Mastering Enterprise JavaBeans and the Java 2 Platform, Enterprise Edition.* John Wiley & Sons, 1999.

[3] Jim Farley, *Java Distributed Computing.* O'Reilly, 1998.

[4] David Flanagan, *JavaScript: The Definitive Guide.* O'Reilly, 1998.

[5] Gall U., Hauck F.J., Promondia: A Java-Based Framework for Real-time Group Communication in the Web, *Proceedings of the Sixth International World Wide Web Conference (1997).*

[6] Land's End web site, http://www.lansdend.com.

[7] McKinley P.K., Malenfant A.M. and Arango J.M. Pavilion: A Middleware Framework for Collaborative Web-Based Applications, in *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work, pp. 179-188, Phoenix, Arizona, November 1999.*

[8] Richard Monson-Haefel, *Enterprise JavaBeans.* O'Reilly, 1999.

[9] Open Buying on the Internet (OBI) V2.0 Technical Specification, OBI Consortium, http://www.openbuy.org, 1999.

[10] Rob Pooley, Perdita Stevens, *Using UML. Software engineering with objects and components.* Addison Wesley Longman Limited, 1999.

[11] Shirmohammadi S., Oliveira J.C. and Georganas N.D., Applet-Based Telecollaboration: A Network-Centric Approach, *IEEE Multimedia Magazine,* Volume 5, Number 2, (April-June 1998), pp.64-73.

[12] UML Notation Guide version 1.1, OMG ad/97-08-05 1997.