University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

2003

E-Commerce Broker Prototype Implementation and Investigation

Rohini Krishnapura
University of Nebraska-Lincoln, rohini@cse.unl.edu

E-Commerce Broker Prototype Implementation and Investigation

Rohini Krishnapura
Computer Science and Engineering
University of Nebraska–Lincoln
Lincoln, NE 66588-0115
rohini@cse.unl.edu

Technical Report TR-CSE-UNL-2003-7
January 2003

Abstract

Personalization has become a popular solution to today's Ecomerce challenges. Various personalization techniques have been researched and marketed. But, one technique may not suit all businesses. What is required is a mechanism to enable different policies based possibly on different personalization techniques. The Ebroker architecture presented here provides a mechanism to enable different policies with minimal effort. We present here the various components of the architecture as well as the features that the architecture provides. The details of a prototype design and implementation are also discussed.

1. Introduction

With the economic slowdown, companies have begun to reinvent the way business is carried out. Ecommerce, which has been particularly affected with the economic slowdown, has been posed with new challenges. These include attracting customers, converting browsers to buyers, maintaining customer loyalty, and preventing customer defection. A study by Forrester Research detailed in [3] claims, 70 percent of all E-commerce sites convince less than 2% of their visitors to buy and nearly 80 percent of retailing newcomers (those selling online for less than 18 months) turn less than 2 percent of their browsers into buyers. 62 percent of the seasoned retailers were found to fare no better.

Retaining customers has been a challenge in business from times immemorial. A report on customer defection [6] claims that companies can boost profits by almost 100% by retaining just 5% more of their customers. In Ecommerce, the effects of losing customers might be proportional but the challenge of retaining them is higher. This is due to the inherent nature of how business is conducted on the Internet. The absence of human intervention by which one can counteract the customer's disinterest is a vital aspect. Customer defection at the shopping cart level, which has become popularly known as the case of the abandoned shopping carts has become a threat to online business. The study detailed in [1] claims that, 65 percent of online consumers bail out and abandon the shopping carts before the final purchase transaction takes place.

Personalization has become one of the answers to these challenges as business researchers claim that a personalized experience leads to more sales. Personalization is about matching content with users' interests or targeting content based on the services the vendor wants to provide. Existing online personalization systems work by constructing customer profiles and using these profiles to provide a personalized experience. These systems are reactive in nature

to the user's interest. Moreover, these systems have business policies embedded within the system making them less flexible to policy changes. In this paper, we propose an architecture to detect the customer's behavior and provide ways of pro-action so as to retain the customer's interest in the transaction. In some ways, the Ebroker is analogous to a sales agent in a person-to-person transaction that is able to detect when a customer is losing interest and take pro-action to close the sale.

The goal of this project is to provide a flexible and scalable mechanism that is independent of policy. It provides support to implement various Ecommerce policies with minimal effort. These goals are realized through the Ebroker architecture. This paper makes contributions in the following areas: we present an architecture with a mechanism for personalization that is independent of policy; we also present the implementation details of a prototype. The prototype makes use of dynamic instrumentation with user monitoring code, which offers flexibility at various levels.

Section 3 details the Ebroker architecture while Section 4 discusses the design of the prototype. Section 5 illustrates the implementation of the Ebroker prototype. Finally, Section 6 presents our conclusions and describes future work. The following section, however, presents related work on personalization systems in detail.

2. Related Work

Existing personalization systems fall into two main categories: rule-based and recommendation systems. Rule-based systems personalize content based on defined business rules. For example, if it was detected that a particular product was surplus in stock, policies can be defined to offer discount on the product when a customer added something to his shopping cart. These rules are independent of the customer's interest but still fall within the definition of personalization since web content is personalized based on the services the vendor wants to provide. Examples of existing rule-based systems are those provided by Broadvision [2], and Vignette [10].

Recommendations systems are based on matching content with users' interests. Two popular techniques are currently being used: collaborative filtering to cluster user's interests and data mining to cluster users based on similar navigation paths. Collaborative filtering [8] to cluster users of similar interests can be done either by explicit ranking by the user or through implicit inference. Much work has also been done on user navigation pattern discovery based on web logs [7] [9]. In both these techniques, users' profiles are constructed and these profiles are used to group users into categories. Amazon's " people who bought this book also bought" feature makes use of this technique of collaborative filtering. Although this technique has its merits, it fails to detect the user's current interest as opposed to his past behavior. If, for example, you were buying a book for a friend, the personalization technique would recommend books based on this. The next time you were buying books for yourself, the personalization system would recommend books based on your previous purchase.

The eGlue Server described in [4] provides an architecture that removes this drawback by constructing dynamic user profiles. The user's current interest is combined with his past

behavior to generate hints and predict the user's next action. The next action is then cached to improve the response time. Business logic is embedded within the system to generate hints.

But, which personalization is best for your business? The success of a personalization technique is inherent to the nature of the online business. Recommendation systems, which have been effective for amazon.com, have not been so successful with Levis Stylefinder [5]. Rule-based systems, which are easier to setup, are best for small companies and not for large-scale retailers. Making recommendation based on collaborative filtering or personalization based on defined business rules are affected by the policies involved within a personalization system. They should not affect the mechanism with which these policies are used to personalize web content. What is required is a mechanism or an architecture that is independent of policy. The policy could be recommendation based, rule-based, inference-based or based on any other new technique. The Ebroker technique offers an architecture that satisfies just this. It provides a scalable and flexible to support maximum policies with minimal effort. Several features like dynamic instrumentation with user monitor code helps to capture user behavior as the web page is being viewed. An event is triggered in response to a specific behavior from the user like scrolling or clicking a link. The actions to these events can be dynamically adjusted based on the policy. They can be used to generate personalized responses or can be geared towards preventing customer defection.

3. Ebroker Architecture

The Ebroker architecture is mainly aimed towards providing a mechanism that is independent of policy. The architecture has several components, which are discussed below. In addition, an outline of the features that are enabled as a result of this architecture is presented.

User Behavior Monitoring

The architecture captures user behavior in the form of events. The various behavioral patterns captured are: clicking on a text link or an image link, navigating the mouse outside the browser area a certain number of times, having a small screen size, clicking on the scroll bar a specific number of times, staying on a particular page for a certain amount of time, abandoning the page within a specific time, aborting the download of an image, and aborting entire page download itself. These are some of the events we are capturing with the implemented prototype although we can capture some more events like form fields, form field completion times, back and refresh buttons, drop down box selections, etc. Since these behaviors are all captured at the client side when the user is actually navigating through the Ecommerce site, the data that we gather is more current as compared to his past purchase behavior or his past navigation path. This behavioral data can be used to infer whether the customer is interested in buying or is losing interest within the Ecommerce site.

Dynamic Instrumentation

The Ebroker's architecture involves dynamic instrumentation of web pages with user behavior monitoring code on the client side. Instrumentation involves inserting hooks or additional code into the web page content. In the Ebroker's context, instrumentation refers to the insertion of the behavior monitoring code into the web page that is generated from the web server. Instrumentation is dynamic since it is done on the fly before the web content reaches the client. In addition, instrumentation of the web page after it is generated from the web

server enables the Ebroker architecture to support dynamic web content. This is essential as current Ecommerce sites are popularly based on dynamic web content rather than static web pages.

Event-action Trigger Mechanism

When the user at the client side exhibits a specific behavioral pattern, an event is triggered. The action is the Ebroker's response to this triggered event. The action to this event is based on the business policy that is specified at that instant. For example, if the user is interested in a web site and is reading a lengthy page by scrolling, this behavior can be detected from the embedded code within the page. An event is triggered when this behavior is detected. At this specific instant, if a rule-based policy was used, and free shipping on a particular product was specified, this would be used to personalize the web content. This personalized content would be provided to the user as the action to the event. The event-action trigger mechanism enables the architecture to be policy independent as the actions can be dynamically modified based on the specified policy. Since the action corresponding to a particular event can be changed dynamically, for the same event triggered at time t_1 and t_2, different policies could be enabled. This also makes the system less predictable and less prone to the users guessing the policies available to them.

Features

The uniqueness of this architecture enables several features such as: easy setup and short customer behavior learning time. Easy setup comes from the concept of dynamic instrumentation. Since the web content that is at the web server side is unaffected, no pre-coding at the server side is required to generate personalized web content. Customer behavior learning time is the time required by the personalization system to learn about the customer's behavior. Since only session behavior is observed, this learning time is small. This is advantageous since the system does not require repeated visits from the customer to learn about his behavior. The personalization system can also be used for a new customer using only his sessional behavior.

Another important attribute provided by this architecture is flexibility. Flexibility itself is at different levels: in terms of Ebroker location, changing actions based on different policies, and action handling location. In concept, the Ebroker architecture can be implemented in either the client side, between the client and the ecommerce server, within the web server, or even behind the web server. The policy independence feature provides the ability to dynamically change policies. Section 5 discusses the policy changer GUI to dynamically change actions based on changing business policies. Flexibility of action handling location arises since the action can be implemented at three locations. The action can be content obtained from the web server, the Ebroker by issuing HTTP requests from the client side or the action can be embedded within the web page during instrumentation providing different levels of transparency. This specifies the transparency from the company perspective, as the policies are not visible to the public. The user is provided the transparency that his behavior is implicitly monitored. Specific ranking of web content is not necessary and there is nothing to disrupt his normal navigational behavior. To preserve his privacy, only session ids are utilized to enable the user to remain anonymous. Cookies are not used to retain any client-side information and the user can be less worried about protecting his privacy.

4. Ebroker Prototype Design

This section describes the design issues that came up during the prototype implementation and the assumptions we made to resolve them. We also describe the various options that we had to make a design decision, the choice that we finally made and the reason for it.

Identifying the WebPages to be intercepted for instrumentation

The vendor could specify a list of URLs, identifying the web pages to be instrumented. Tags could then be used to distinguish the web pages that need to be instrumented from the ones that don't need to be. Although this would work for static web content, dynamic pages could have the same URLs, so the problem of uniquely identifying a page would arise. Moreover, this solution increases the specifications required by the vendor to setup the Ebroker. We needed to make the setup of Ebroker with minimum requirements from the vendor. A specific "NO_INST" tag could be inserted on top of pages while dynamically creating them or have to be inserted manually if they're already static to indicate that instrumentation for these pages is not required. Pdf(s), images, doc(s), etc. can be filtered out by looking at the "text/html" tag in the HTTP header. For the prototype, all vendor text/html pages except the action pages will be intercepted and instrumented. If more granularity is required, the vendor has to specify the list of URLs and distinguishing between dynamic web pages with same URLs need to be considered. A better solution would be obtain a product catalog and identify the locations within this catalog to be instrumented. The use of a product catalog is described in detail in Section 6 as future work.

Identifying the events to be instrumented

To identify which events have to be instrumented in which pages was another issue. Here again, the obvious solution was to extend the URLs to be instrumented specification sheet with the corresponding events to be captured. A better choice would be identify the events along with the product catalog.

For the prototype, while integrating with an experimental Ecommerce site, all pages were instrumented with all events.

Policy Specification

An event-action table was designed to specify the correspondence between the events and actions. The business policy specified at that instant had to be translated to a specific action, which would be taken in response to an event. For the prototype, a policy changer was implemented which would enable the vendor to change the action on any event based on the change in the business policy. In concept, any external process could dynamically change the actions. The external process itself could be a policy engine geared towards finding the optimal business policy and change the actions based on this algorithm.

5. Implementation

For the prototype, the Ebroker was implemented by integrating it in the dispatcher of the LSQoS described in [11]. This enabled us to easily implement dynamic instrumentation by gathering all the packets that corresponds to a particular web page. The data within these packets was extracted and the behavior-monitor code was inserted before forwarding these packets to the client. An experimental Ecommerce web site was then setup and integrated

with the Ebroker prototype. Initially, all pages had all events instrumented to capture user behavior. We first describe here the changes that were made to the LSQoS code to implement the Ebroker, which is then followed by details unique to the Ebroker prototype implementation. This includes the implementation of the log analyzer as well as the policy changer GUI.

Capturing the web page

One of the major changes made to the LSQoS was to store the data in the packet as it is captured from the web server till the entire web page is constructed. The TCP protocol was also extended to send acknowledgements to the server independent of those received from the client. This prevented retransmissions from the web server while waiting for acknowledgement, as packets were not transmitted to the client till the entire web page was constructed. The req_st structure that is used in the LSQoS was extended to enable the provide information for the Ebroker logging such as user agent, referrer, the status message returned from the web server along with the status code, the response line that can be obtained from the first response packet from the web server, etc. Fig. 1 shows the extended structure. The eTag is used to indicate whether the web page is to be captured and then instrumented. If this is set, only then is the data in the packet buffered to construct the web page.

```
typedef struct req_st req;
struct req_st
{        time_t              arrive_time;
         bool                conn_close;
         uint32_t            dtoc_start_seq;
         uint32_t            c_start_seq;
         uint16_t            req_num;
         float               req_file_priority;
         unsigned char       *req_data;
         int                 req_size;
         char                *req_line;
         ulong               remotehost;
          int                response_size;
         int                 status;
         char                *status_msg;
         char                *response_line;
         char                *user_agent;
         char                *referrer;
         char                *ebrokerFile;
         bool                eTag;
         bool                dynamicPage;
         int                 eventIndex;
         req                 *next;
};
```

Fig. 1

The dynamicPage field is used to support dynamically generated web content. In HTTP 1.1, dynamically generated data is sent as packets with the HTTP header containing "Transfer-Encoding: chunked" field to indicate that data is chunked. Packets with chunked data do not have the "Content-Length field" in the HTTP header. Since the Ebroker code and the basic LSQoS code keeps a track of this to detect the last packet, the packets had to be "dechunked" to obtain the chunk size. This meant examining the data within the packet to detect a chunk size and gathering all the chunks based on this size. The last chunk could be indicated with a

specific terminating sequence as specified in HTTP 1.1. The dynamicField is used to indicate that dechunking is required.

Implementation of Dynamic Instrumentation

The captured web page is then instrumented with event capture code. The event capture code is implemented in JavaScript. The HTTP_request structure described above has an "eTag" field, which is indicated with the tag "TAG-I" to indicate that instrumentation is to be done, and "TAG_NI" to indicate that no instrumentation is required. As discussed in Section 4, all text/html web content was tagged for instrumentation. The type of data is gathered by examining the field "Content-Type: " in the HTTP header of the packet. Once the entire web page is captured, instrumentation process is started. When the instrumentation is over, the entire page is broken into packets and sent to the client in appropriate packet sizes. The TCP sequence of the packet that is sent is adjusted to indicate the extra data added by the instrumentation process so that the client honors the packets.

Event Action Table

The Event Action Table is the data structure through which the event-action trigger mechanism is implemented. The table is indexed into every time an event is triggered, to determine the action to be taken. In addition, this table is looked up during instrumentation to access the behavior capture code to insert. The structure of the table is given in Table. I.

Event	Capture Code	Action Type	Action	Action Code

Table. I

The capture code points to a library where the capture code is defined. The action type indicates where the action is to be handled. If the action is handled within the client, the action code is embedded during the instrumentation phase so that the action is triggered appropriately in response to the event. The action entry in the event-action table specifies the action that is to be instrumented. If however, the action content is to be retrieved from the Ebroker or the web server, the action type will indicate this and when the event is triggered, the action field indicates the action content to be requested. The action code is a pointer to a function where the action content is requested.

Policy Changer GUI

The event action table is also the data structure through which the policies are translated into actions. The entire table is stored in shared memory so that actions can be updated in accordance with changing policies by external processes. For the prototype, we implemented a Policy Changer GUI through which the actions can be changed as policies change. The policy changer accesses this shared memory to update the Event-Action table to change the appropriate fields. The policy changer can also specify the Ebroker not to take any action for an event through this mechanism. Semaphores are used for synchronization between the Ebroker and the GUI accessing the Event-Action table at the same time. Conceptually, the Policy Changer can be extended to change policies and hence actions, based on an algorithm which maximizes returns. We discuss this in future work. Fig. 2 shows the tree structure of the events, and the actions that can be taken when these events are triggered.
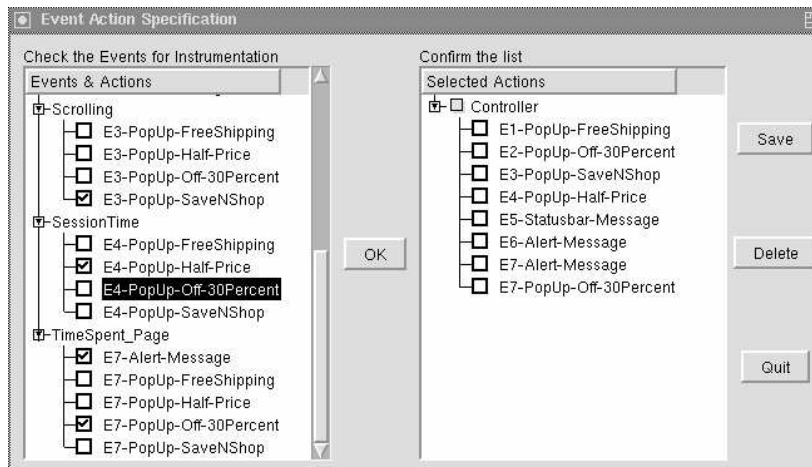
Fig. 2

Event-Action Logging

A field in the Ebroker configuration file enables logging. The Ebroker log captures the request-reply associations. In addition, the log captures the event that was captured and the action that was taken. The format of the Ebroker log is as given in Table. II. The format is in compliance with the extended log format as specified by the w3 consortium except for two additional fields. These two extra fields are Event Triggered and Action Taken, which are relevant to the Ebroker architecture. In fact, it is these two fields in conjunction with the other fields that help us determine various statistics from the Ebroker log. The Ebroker Online Log analyzer describes in the next section uses this log to display these statistics.

Host	User Login	Time of Request	HTTP Request	HTTP Response Status	Bytes Transferred	Referrer	User Agent	Event Triggered	Action Taken

Table. II

The Host field in the log specifies the host machine of the web client, which is issuing the HTTP request. The user login is used to log authenticated users. Although this field is specified in the log, this is not logged by the Ebroker to allow the users to remain anonymous. The time of request is the time at which Ebroker gets the request packet from the client. The HTTP response status is the return code from the Web Server in response to the request. The Referrer field indicates from which link the user was previously viewing. The User Agent field indicates the type of browser on the client side. The Event Triggered specifies if a particular event was triggered from the current page and the Action Taken specifies the action that was taken in response to the event.

Ebroker Log Analyzer

The functionality of the Ebroker Log Analyzer was based on a GNU licensed tool awstats 4.1. The tool was redesigned to collect information based on the Ebroker log structure and evaluate statistics that are more relevant to the Ebroker. A perl script was used to display the

statistics obtained from the analysis. The GUI opens a client socket to obtain the Ebroker log which is then used to extract information such as inter-action time, the number of times a particular event was triggered, the events triggered by a specific user and the corresponding actions to these events, and the time at which an event was triggered. The GUI is also pseudo real-time as fresh updates of the log are obtained every 50 seconds from the Ebroker system to analyze and display the statistics. Fig. 3 shows the events that have occurred for a particular user, the time at which the event was triggered and the actions that have been taken based on the policy at that time. The number of times an event has been triggered in a specific time that's set in the GUI configuration file is also displayed.
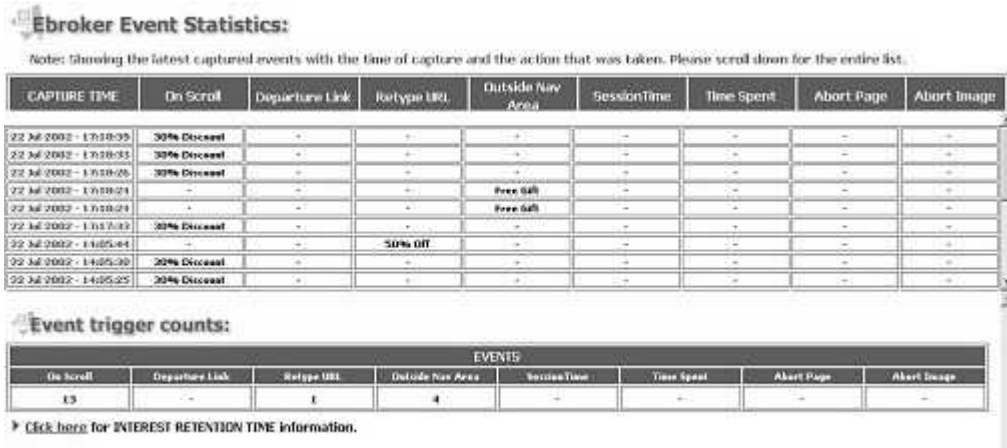

Fig. 3

6. Conclusions and Future Work

We have presented in this paper an architecture to implement ecommerce policies with minimal effort. The mechanism was presented though which policies can be translated to action in response to triggered events. The future work involves finding a method to evaluate the Ebroker mechanism independent of the policy. Also, the current method of storing the entire page before the instrumentation phase could be time-consuming. A method of stream parsing or even instrumenting packets with event capture code needs to be researched. Scalability of the Ebroker architecture also needs to be evaluated to a wide variety of events and also with a large number of clients. Classifying behavioral patterns and associating relative weights to events to give a qualitative measure to user's interest or disinterest is also an area we will focus on in the future. The significance of the capture behavioral pattern needs to be examined.

References

[1]   Boston Consulting Group, April 17, 2000 - Online Retailing In North America Reached $33.1 Billion In and is projected to top $61 billion in April 2000, http://www.bcg.com/new_ideas/new_ideas_subpage1.asp.

[2]   Broad Vision – http://www.broadvision.com.

[3]   Cooperstein, D., Delhagen, K., Aber , A., Levin , K., Making Net Shoppers Loyal, *Forrester Report*, June 1999.

[4] Datta, A., Dutta, K., VanderMeer, D., Ramammritham, K., Shamkant, N., An architecture to support scalable online personalization on the web, *Proceedings of the 2nd ACM Conference on Electronic Commerce (EC'00),* October 17-20, 2000, Minneapolis, Minnesota.

[5] Lighthouse, Personalisation goes one-on-one with reality, August 2000, http://www.shorewalker.com/hype/hype60.html.

[6] Reichheld, F., Sasser, E., Zero defections: Quality comes to services, *Harvard Business Review*, September-October 1990.

[7] Shahabi, C., Knowledge discovery from user's web-page navigation, *Proceedings of RIDE'97 – Seventh International Workshop on research issues in data engineering*, 1997.

[8] Shardanand, U., Maes, P., Social information filtering: algorithms for automating "word of mouth", *Proceedings on Human Factors in Computing Systems*, May 1995.

[9] Spilioupoulou, M., Faulstich, L.S., and Winkler, K., A data miner analyzing the navigational behavior of web users, *International conference of ACAI'99: Workshop on Machine Learning in User modeling*, 1999.

[10] Vignette, http://www.vignette.com.

[11] Wei, C,. A QoS assurance mechanism for Cluster-based web servers, Masters Degree Report presented to UNL, November 2000.