Bulletin of the Technical Committee on

Data Engineering

March 2000 Vol. 23 No. 1

IEEE Computer Society

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering (http: www. is open to all current members of the IEEE Computer Society who are interested in database systems.

The web page for the Data Engineering Bulletin is http://www.research.microsoft.com/research/db/debull. The web page for the TC on Data Engineering is http://www.ccs.neu.edu/groups/IEEE/tcde/index.html.

Letter from the Editor-in-Chief

Election Results for the Chair of the TC on Data Engineering

The TC on Data Engineering held an election for TC chair. Below is the text of a message from NIchelle Schoultz reporting on the results of the election. I would like to congratulate Betty Salzberg on her re-election as TC chair.

```
Dear Members of the Technical Committee on Data Engineering:

Betty Salzberg has been elected as the 2000 Chair for TCDE. She was elected by a
unanimous decision, with a total of 8 members voting. If you have any questions,
please feel free to contact me nschoultz@computer.org
Thank you.

Nichelle Schoultz
Volunteer Services Coordinator
```

About the Current Issue

The world of the world wide web is undergoing explosive growth. An important element of that growth is the emergence of electronic commerce as a significant factor in the global economy. Indeed, while it is sometimes hard to understand the stock market, the valuations of companies that have a presence in the e-commerce sector have exploded, while the stock market valuations of more traditional companies have languished. It does not take a genius to understand that e-commerce is having and will continue to have an enormous impact the world economy.

Elke Rundensteiner has assembled a collection of articles for this issue that emphasizes the database technology role in e-commerce. Included in the issue are papers from the academic community, industrial research, and industry. The issue provides a valuable snapshot on what is going on in this rapidly changing world. I'd like to thank Elke for her fine choice of topic and her very successful effort on this issue. I want to thank her as well for her efforts throughout her tenure as an editor of the Bulletin.

New Editors

Careful readers will note that the inside front cover now includes our two new editors, Sunita Sarawagi and Alon Levy. If you have suggestions for the remaining two editorial positions, please send me email at lomet@microsoft.com.

David Lomet
Microsoft Corporation

Letter from the TCDE Chair

The ICDE 2000 Conference was held February 29 to March 3 in San Diego. The attendance was 217. The technical program was praised by many; only 41 of nearly 300 submitted papers were accepted. In addition to the technical papers, several panels, industrial sessions, demonstrations and poster sessions took place. Highlights included keynote addresses by Jim Gray of Microsoft, the 1999 Turing award winner, on ”Rules of Thumb in Data Engineering” and by Dennis Tsichritzis, head of the GMD [German] National Research Center for Information Technology on ”The Changing Art of Computer Research.”

The best student paper was awarded to V. Ganti, J. Gehrke and R. Ramakrishnan (Univ. of Wisconsin, Madison) for ”DEMON: Mining and Monitoring Evolving Data.” The best paper award went to S. Chaudhuri and V. Narasayya from Microsoft for ”Automating Statistics Management for Query Optimizers.”

I would like to thank the organizers for working so hard to produce a high-quality technical conference. In particular, I thank Paul Larson, the general chair and David Lomet and Gerhard Weikum, the program chairs. In addition, the organizing committee members and the program committee members all contributed to making this conference an unqualified success.

The next ICDE will be held in Heidelberg, Germany in April 2001. I hope that you will all plan to attend. It was also announced that the 2002 conference will be in Silicon Valley, California.

The TCDE held an open meeting during the ICDE conference luncheon. These meetings take place yearly at the ICDE and all TCDE members are invited to attend. One item discussed at this meeting was the participation with ACM SIGMOD in the SIGMOD anthology. This is a collection of CDROMs being made available to SIGMOD members and TCDE members. The SIGMOD members have received the first 5 disks. We are arranging to mail to ICDE members next fall all the disks (probably about 13) which will be available at that time. These disks contain proceedings of ICDE, SIGMOD, VLDB,EDBT, PDIS and ICDT conferences and ACM PODS and IEEE TKDE journals and the Data Engineering Bulletin, among other publications in the database systems area.

Betty Salzberg
Northeastern University

Letter from the Special Issue Editor

The Internet has taken over our world in a storm, affecting most aspects of our personal as well as professional lives. One of the most remarkable phenomenon is how the internet has influenced business, both in terms of how we conduct business nowadays as well as in terms of opening up novel business opportunities. This has resulted in a surge of start-up companies filling this new nitch of business opportunities, as well as established companies re-inventing themselves by reaching out to their business partners (business-to-business e-commerce) and their customers (business-to-customer e-commerce) in new ways as well as internally streamlining their own business processes.

In this issue, we now take a look at *Database Technology in E-Commerce*. We study new requirements that E-commerce applications are posing to database technology (for example, on database design); new tools being developed for E-commerce applications (for example, XML/EDI tools), as well as presenting examples of working systems built using these new tools.

The first paper "Data Mining Techniques for Personalization" by Charu C. Aggarwal and Philip S. Yu provides an overview of techniques used to tailor a system to specific user needs. Their survey compares in particular collaborative filtering, content based methods, and content based collaborative filtering methods.

A natural complement to this personalization issue is the work by Kajal Claypool, Li Chen, and Elke A. Rundensteiner on "Personal Views for Web Catalogs". They illustrate how user preferences or behavior extracted with one of these mining techniques can be utilized by companies to better serve their customers (and thus themselves) by now offering and maintaining personal views to E-commerce catalogs.

The next paper by M. Tamer Özsu and Paul Iglinski also focusses on electronic catalogs, The paper titled "An Interoperable Multimedia Catalog System for Electronic Commerce" introduces the authors's project on developing the infrastructure of multimedia catalogs build over possibly distributed storage systems.

The paper by Il-Yeol Song and Kyu-Young Whang entitled "Large Database Design for Real-World E- Commerce Systems" outlines issues faced when designing a database for e-commerce environments based on their experience with real-world e-commerce database systems in several domains, such as an online shopping mall, an online service delivery, and an engineering application.

The paper "End-to-end E-commerce Application Development Based on XML Tools" by Weidong Kou, David Lauzon, William O'Farrell, Teo Loo See, Daniel Wee, Daniel Tan, Kelvin Cheung, Richard Gregory, Kostas Kontogiannis, and John Mylopoulos describes how newly emerging XML tools can assist organizations in deploying e-commerce applications. An e-business application framework and architecture are introduced using a virtual store scenario.

Serge Abiteboul, Sophie Cluet, and Laurent Mignet put forth that "Declarative Specification of Electronic Commerce Applications" could allow us in the future to develop typical E-commerce applications much more rapidly. The ActiveView language is sketched as possible candidate for such specifications.

Manish Gupta from Excelon Inc (formerly Object Design Inc.) discusses design considerations of a high performance high load e-commerce web system. The paper "TPC-W E-Commerce Benchmark Using Javlin/ObjectStore" includes preliminary experimental results on the TPC-W benchmark.

Lastly, Ee-Peng Lim and Wee-Keong Ng in "An Overview of the Agent-Based Electronic Commerce System Project" outline issues they have encountered when building business-to-consumer e-commerce systems using agent technologies.

Elke A. Rundensteiner
Worcester Polytechnic Insitute

Data Mining Techniques for Personalization

Charu C. Aggarwal, Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

This paper discusses an overview of data mining techniques for personalization. It discusses some of the standard techniques which are used in order to adapt and increase the ability of the system to tailor itself to specific user behavior. We discuss several such techniques such as collaborative filtering, content based methods, and content based collaborative filtering methods. We examine the specific applicability of these techniques to various scenarios and the broad advantages of each in specific situations.

1 Introduction

In recent years, electronic commerce applications have gained considerable importance because of the importance of the web in electronic transactions. The Web is rapidly becoming a major source of revenue for electronic merchants such as Amazon, and Ebay, all of which use some form of personalization in order to target specific customers based on their pattern of buying behavior.

The idea in personalization in web business applications is to collect information about the customer in some direct or indirect form and use this information in order to develop personalization tools. Thus, an electronic commerce site may continue to keep information about the behavior of the customer in some explicit or implicit form. The two typical methods for information collection at an E-commerce site are as follows:

- **Explicit Collection:** In this case, the customer may specify certain interests, which reflect his desire of buying certain products. Example of such methods include collaborative filtering methods in which customers specify ratings for particular products and these ratings are used in order to make peer recommendations.

- **Implicit Collection:** In implicit collection, the E-commerce merchant may collect the buying behavior of the customer. In addition, the browsing or other bahavior may also be captured using the web or trace log. There are several limitations to the nature of implict collection because of privacy requirements of users.

This paper is organized as follows. In the next section, we will discuss collboarative filtering which is the technique of using the explicit information in order to make peer recommendations. In subsequent sections, we will discuss more indirect ways of using data mining techniques on the customer behavior in order to capture his patterns. Finally, we will discuss an overview of the general applicabilities of these techniques to different scenaries.

2 Collaborative Filtering Techniques

In collaborative filtering techniques, the idea is appropriate for e-commerce merchants offering one or more groups of relatively homogeneous items such as compact disks, videos, books, software and the like. Collaborative filtering refers to the notions of multiple users "sharing" recommendations in the form of "ratings", for various items. The key idea is that collaborating users incur the cost (in time and effort) of rating various subsets of items, and in turn receive the *benefit* of sharing in the collective group knowledge. For example, they can view predicted ratings of other items that they identify, see ordered lists of items whose predicted ratings are the highest and so on. The explicit ratings entered by the user are typically done so in some kind of user interface which we will refer to as a profiler dialog.

The importance of collaborative filtering arises from the fact that the number of possible items at an E-commerce site is far greater than the number of items that a user can be expected to rate in a reasonable amount of time. In fact, the catch in collaborative filtering is to give users enough motivation so that a sufficient number of items are rated. Typically, it is known that given the choice, only $1 - 2\%$ of the users are likely to rate items, and even among them only a small number of items can be rated. In order to implement a site in which every user rates items, E-commerce merchants typically provide incentives such as free email [14] or other products, which may then be used to elicit greater user behavior. Once a greater level of user behavior has been achieved, the ratings of various items may be predicted for a given user by creating peer groups. In this case, the idea is to construct groups of users having a similar behavior and using their *aggregate* behavior in order to predict the ratings for a given user-product combination which has not been explicitly specified in a profiler dialog [10, 12]. Recent techniques have also been developed[3] which build peer groups in an even more general way by using groups of people which have either positively or negatively correlated behavior.

Some examples of useful questions which could be resolved by a recommender system are as follows:

- Show user i the projected rating for item j, assuming that the item has not already been rated by that user and assuming that the algorithm is able to meaningfully estimate the item's rating. If the user has already rated the item, that item is shown instead.

- Show user i an ordered list of up to M (as yet unrated) items from a subset $\mathcal{I}$ which are projected to be liked the most, subject to the constraint that each of them has a projected rating of at least some minimum value r. This subset may be chosen implicitly or explicitly in a number of ways. For example, it may consist of a set of promotional items.

- Show user i an ordered list of up to M (as yet unrated) items from a subset $\mathcal{I}$ which are projected to be liked the least subject to the constraint that each of them has a projected rating of at most some maximum value r.

- Select an ordered list of up to M users from a subset $\mathcal{J}$ each of which has a projected rating of at least r that are projected to like item j the most. This is a query which is useful to the E-commerce merchant in order to generate mailings for specific sets of promotion items.

Even though collaborative filtering is a very effective technique, its shortcomings are clear; it is difficult to collect a sufficient amount of user feedback to begin with. For such cases, it may be more desirable to pick techniques which deal with data corresponding to actual user behavior. Such methods are referred to as content based techniques.

3 Content Based Systems

In content based systems, the feedback of other users is not relevant while deciding the recommendations for a given user. The idea is to characterize the user behavior in terms of his *content* of accesses; a new attribute

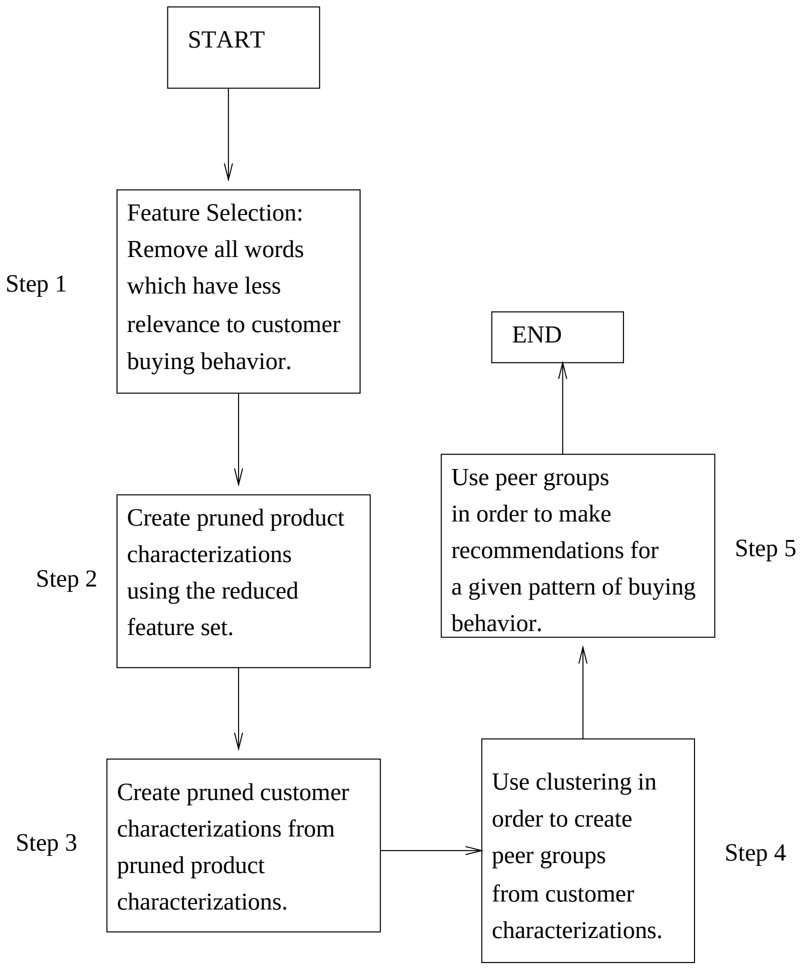

START
Step 1
Feature Selection: Remove all words which have less relevance to customer buying behavior.
Step 2
Create pruned product characterizations using the reduced feature set.
Step 3
Create pruned customer characterizations from pruned product characterizations.
Step 4
Use clustering in order to create peer groups from customer characterizations.
Step 5
Use peer groups in order to make recommendations for a given pattern of buying behavior.
END

Figure 1: Content Based Collaborative Filtering

system corresponding to user behavior. For example, while tracking the user behavior at an E-commerce site, the content of his behavior may refer to the set of text documents which have been accessed by him.

At the same time, the set of products at an E-commerce site may be categorized or clustered based on the content of the products. For a given user, one may then categorize his behavior by using IR similarity techniques in order to compare the content of the cluster and the content of user access patterns [4]. The results of this comparison may be used in order to make product recommendations. The idea here is to create a content taxonomy and characterize each user by the content category of the products that he purchased. Similar techniques may be used in order to make web page recommendations by analyzing the web pages which were browsed.

Content based techniques have the advantages of directness, simplicity, and redundancy of user feedback, but they lack the sophistication of collaborative filtering systems which analyze the behavior of entire peer groups for making recommendations. For this purpose, it is useful to examine and analyze the behavior of content based collaboratuve systems which do not require user feedback; yet use the concept of user peer groups for making recommendations.

4 Content Based Collaborative Filtering Systems

In this section, we will discuss an application of clustering in order to build content based collaborative filtering systems. Content based collaborative filtering systems are relevant in providing personalized recommendations at an E-commerce site. In such systems, a past history of customer behavior is available, which may be used for making future recommendations for individual customers. We also assume that a "content characterization" of products is available in order to perform recommendations. These characterizations may be (but are not restricted to) the text description of the products which are available at the web site. The key here is that the characterizations should be such that they contain attributes (or textual words) which are highly correlated with buying behavior. In this sense, using carefully defined content attributes which are specific to the domain knowledge in question can be very useful for making recommendations. For example, in an engine which recommends CDs, the nature of the characterizations could be the singer name, music category, composer etc., since all of these attributes are likely to be highly correlated with buying behavior. On the other hand, if the only information available is the raw textual description of the products, then it may be desirable to use some kind of feature selection process in order to decide which words are most relevant to the process of making recommendations.

We will now proceed to describe the overall process and method for performing content-based collaborative filtering. The collaborative filtering process consists of the following sequence of steps, all of which are discussed in Figure 1.

(1) **Feature Selection:** It is possible that the initial characterization of the products is quite noisy, and not all of the textual descriptions are directly related to buying behavior. For example, *stop words* (commonly occuring words in the language) in the description are unlikely to have much connection with the buying pattern in the products. In order to perform the feature selection, we perform the following process: we first create a customer characterization by concatenating the text descriptions for each product bought by the customer. Let the set of words in the lexicon describing the products be indexed by $i \in \{1, \ldots, k\}$, and let the set of customers j for which buying behavior is available be indexed by $j \in \{1, \ldots, n\}$. The frequency of presence of word i in customer characterization j is denoted by $F(i, j)$. The fractional presence of a word i for customer j is denoted by $P(i, j)$ and is defined as follows:

$$P(i,j) = \frac{F(i,j)}{\sum_{j \in \text{All customers}} F(i,j)} \tag{1}$$

Note that when the word $i = i_0$ is noisy in its distribution across the different products, then the values of $P(i_0, j)$ are likely to be similar for different values of j. The *gini index* for the word i is denoted by $G(i)$, and is defined as follows:

$$G(i) = 1 - \sqrt{\sum_j P(i,j)^2} \tag{2}$$

When the word i is noisy in its distribution across the different customers, then the value of $G(i)$ is high. Thus, in order to pick the content which is most discriminating in behavioral patterns, we pick the words with the lowest gini index. The process of finding the words with the lowest gini index is indicated in Step 1 of Figure 1.

(2) **Creating Customer Characterizations:** In the second stage of the procedure, we create the customer characterizations from the text descriptions by concatenating the content characterizations of the products bought by the individual consumers. To do so, we first prune the content characterizations of each product by removing those features or words which have high gini index. Then we create customer characterizations by concatenating together these pruned product characterizations for a given customer (Step 3).

(3) **Clustering:** In the subsequent stage, we use the selected features in order to perform the clustering of the customers into peer groups. This clustering can either be done using unsupervised methods, or by supervision from a pre-existing set of classes of products such that the classification is directly related to buying behavior.

(4) **Making Recommendations:** In the final stage, we make recommendations for the different sets of customers. In order to make the recommendations for a given customer, we find the closest sets of clusters for the content characterization of that customer. Finding the content characterization for a given customer may sometimes be a little tricky in that a weighted concatenation of the content characterizations of the individual products bought by that customer may be needed. The weighting may be done in different ways by giving greater weightage to the more recent set of products bought by the customer. The set of entities in this closest set of clusters forms the *peer group*. The buying behavior of this peer group is used in order to make recommendations. Specifically, the most frequently bought products in this peer group may be used as the recommendations. Several variations of the nature of queries are possible, and are discussed subsequently.

We have implemented some of these approaches in a content-based collaborative filtering engine for making recommendations, and it seems to provide significantly more effective results than a content based filtering engine which uses only the identity attributes of the products in order to do the clustering.

Several kinds of queries may be resolved using such a system by using minor variations of the method discussed for making recommendations:

(1) For a given set of products browsed/bought, find the best recommendation list.

(2) For a given customer and a set of products browsed/bought by him in the current session, find the best set of products for that customer.

(3) For a given customer, find the best set of products for that customer.

(4) For the queries (1), (2), and (3) above, find the recommendation list out of a prespecified promotion list.

(5) Find the closest peers for a given customer.

(6) Find the profile of the customers who will like a product the most.

Most of the above queries (with the exception of (6)) can be solved by using a different content characterization for the customer, and using this content characterization in order to find the peer group for the customer. For the case of query (6), we first find the peer group for the content characterization of the current product, and then find the dominant profile characteristics of this group of customers. In order to do so, the quantitative association rule method [13] may be used.

5 Summary

In this paper, we discussed several implicit and explicit techniques for performing personalization at an electronic commerce site. The explicit methods for personalization include user-feedback and ratings, whereas the implicit behavior include the observation of user buying behavior. Each of these techniques have their own drawbacks:

- There are limitations to how much effort a user is willing to put in in order to explicitly provide his preferences using methods such as a profiler dialog. Typical response rates for schemes in which ratings may be provided on a voluntary basis are only $1 - 2\%$. Furthermore, often such information may be too sparse to be of too much use.

- There are limitations to how much one can track user behavior without violating privacy concerns that many people may have. Often, for E-commerce sits which work without the use of a registration process have no way of tracking behavior in an effective and acceptable way.

Given the limits on tracking the online user behavior, it is often easier to collect user behavior at a batch level, since the data collection process is not hindered by the online nature of the user actions. For such cases many data mining techniques such as associations, clustering, and categorization [1, 2, 5, 6] are known.

References

[1] Aggarwal C. C. et. al. "Fast Algorithms for Projected Clustering." *Proceedings of the ACM SIGMOD Conference of Management of Data*, 1999.

[2] Aggarwal C. C., Yu P. S. "Finding Generalized Projected Clusters in High Dimensional Spaces." *Proceedings of the ACM SIGMOD Conference on Management of Data,* 2000.

[3] Aggarwal C. C., Wolf J. L., Wu K.-L., Yu P. S. "Horting Hatches an Egg: Fast Algorithms for Collborative Filtering." *Proceedings of the ACM SIGKDD Conference*, 1999.

[4] Aggarwal C. C., Gates S. C., Yu P. S. "On the merits of building categorization systems by supervised clustering." *Proceedings of the ACM SIGKDD Conference*, 1999.

[5] Agrawal R., Imielinski T., and Swami A. "Mining association rules between sets of items in very large databases." *Proceedings of the ACM SIGMOD Conference on Management of data,* pages 207-216, 1993.

[6] Chen M. S., Han J., Yu P. S. Data Mining: An overview from the database perspective. *IEEE Transcations on Knowledge and Data Engineering*.

[7] www.amazon.com

[8] www.ebay.com

[9] Freund Y., Iyer B., Shapiro R., Singer Y. "An efficient boosting algorithm for combining preferences."*International Conference on Machine Learning,* Madison, WI, 1998.

[10] Greening D., "Building Consumer Trust with Accurate Product Recommendations.", Likeminds White Paper LMWSWP-210-6966, 1997.

[11] Resnick P., Varian H. "Recommender Systems", *Communications of the ACM*, Volume 40, No. 3, pages 56-58, 1997.

[12] Shardanand U., Maes P. "Social Information Filtering: Algorithms for Automating Word of Mouth." *Proceedings of the CHI'95*, Denver CO, pp. 210-217, 1995.

[13] Srikant R., and Agrawal R. Mining quantitative association rules in large relational tables. *Proceedings of the 1996 ACM SIGMOD Conference on Management of Data*. Montreal, Canada, June 1996.

[14] www.hotmail.com

Personal Views for Web Catalogs [*]

Kajal T. Claypool, Li Chen and Elke A. Rundensteiner
Department of Computer Science
Worcester Polytechnic Institute
Worcester, MA 01609–2280
{kajal|lichen|rundenst}@cs.wpi.edu

Abstract

Large growth in e-commerce has culiminated in a technology boom to enable companies to better serve their consumers. The front-end of the e-commerce business is to better reach the consumer which means to better serve the information on the Web. An end-to-end solution that provides such capabilities includes technology to enable personalization of information, to serve the personalized information to individual users, and to manage change both in terms of new data as well as in terms of the evolution of personal taste of the individual user. In this work, we present an approach that allows automated generation of diversified and customized web pages from an object database (based on the ODMG object model), and automated maintenance of these web pages once they have been build. The strength of our approach is its superior re-structuring capabilities to produce *personal views* as well as its generic propagation framework to propagate schema changes, data updates, security information and so on from the base source to the personal view and vice versa.

1 Introduction

Internet technologies and applications have grown more rapidly than anyone could have envisioned even five years ago, with a projection of 127% increase in Web sales for the current year alone (*http:// www.internetindicators.com /facts.html)*. What started out as static personal Web pages with photographs to be shared with friends and family or hastily compiled company brochures has quickly culminated into a myriad of sophisticated hardware and software applications. Companies today are moving beyond static information to provide to their users dynamic and personalized information to create new value for their stake-holders. An end-to-end solution that provides such dynamic capabilities includes technology to enable personalization of information, to serve the personalized information to individual users, and to manage change both in terms of new data as well as in terms of the evolution of personal taste of the individual user.

Consider for example the `eToys` web site (*http: //www.etoys.com*), one of the biggest online retailer of toys for children. Currently, `eToys` provides a minimum two-level navigation of its web site based on *child's age* and *toy type* categorization. Sales on toys or the *hot* toys are shown as separate links and also have an age categorization. A user of the `eToys` site currently navigates individually through each of links. However, a frequent user

[*]This work was supported in part by the NSF NYI grant #IRI 94-57609. We would also like to thank our industrial sponsors, in particular, IBM for the IBM partnership award. Kajal Claypool and Li Chen would also like to thank GE for the GE Corporate Fellowship and IBM for the IBM corporate fellowship respectively.

of the web site often follows a set pattern of links. For example, a user with a `4 year old` would be most likely to look for toys in the age category of `4 to 6`. This user may be better served if the `eToys` web site collated information on sales, hot toys, and toys such as cars for `4 year olds` into one web page. Of course, since this information varies from one individual user to another where one user might target boys and another might target girls, each user would be best served with a *personal view* of the `eToys` database. New toys added into the database should also be reflected in the user's personal view. Moreover, as the individual user's tastes change, the child grows older or the child is no longer interested in cars, the *personal view* needs to be adapted to reflect this change in taste. Given the volume of users, 17 million US households alone (*http://www.nua.ie/surveys*), a key in providing for such dynamic behavior is a high degree of automation of both the view definition as well as the view maintenance process.

Today, technology such as *collaborative* or *agent-based filtering* [AY00] are often used as mechanisms to arrive at the personal set of information. We do not look at these technologies per say but assume that one or more such technologies are available to arrive at the personalized information. We focus instead on actually serving users personalized information (database views with complex re-structuring capabilities) and on managing change on already served pages (a generic propagation framework that handles data updates as well as schema changes) by using and extending existing database (OODB) technology.

Recently research efforts [AMM98, FFLS97, CRCK98] have been made to automate aspects of providing dynamic views. For example, Araneus [AMM98] is an attempt to re-apply relational database concepts and abstractions to generate web sites. Strudel [FFLS97] instead has introduced a hyper-graph data model to capture the web structure and an associated web query language, StruQL. However, while these approaches focus on generating diversified web sites, they do not provide an easy to use mechanism in terms of creating the transformations for building these views. Nor do they look at the issues of maintainability in terms of reflecting the changing profile or taste, or propagating changes or additions to the existing data set.

Our Re-Web [CRCK98] system is an *easy to use, automated* web-site management tool (WSMS) based on the ODMG standard that focuses on building a *personal view* for each user and, in addition, manages evolution as well as data changes of the *personal views*. Re-Web can translate database information to web information, re-structure information at the database level and finally generate web pages from the information in the database. Exploiting the modeling power of the OO model, we have defined web semantics that allow us to map between the XML/HTML constructs and the ODMG object model constructs. At the database level we use a flexible and extensible re-structuring facility, SERF [RCC00] to support complex re-structuring for the creation of new *personal views*. To increase the ease of web site management, Re-Web also exploits the notion of a library of re-usable transformations from SERF [RCC00]. In particular, we define a library of typical Web restructuring and support a visual tool for building the transformations, further simplifying the web restructuring effort.

To address the maintainability of the *personal views* we also propose a generic propagation framework that can handle the propagation of information such as data updates, schema changes, security information from one view to another or from the base information to the view. Thus, using this framework we can tackle in an automated fashion the updatability of *personal views* as and when new data is added to the base or existing data is updated. Moreover, as the user's profile changes either by a manual update to the profile or by some filtering mechanism, the propagation framework can be utilized to evolve the existing *personal view* accordingly. Once a database view is evolved, the adapted web pages are automatically generated by Re-Web.

2 Personal Views

Generating Web Pages: Mapping ODMG to XML. For web site generation, we provide three tiers of web semantics representations. The top tier is the HTML representation, i.e., the web pages themselves, which include some visual metaphors and styles. Ideally the users navigate through the whole web site via its HTML pages in a unique and consistent fashion. The XML representation of the HTML pages forms the middle-layer, an intermediate translation between the ODMG data model is at the bottom tier and the equivalent loss-less representation of the HTML web pages.

Figure 1: The `eToys` Home Page.

Figure 2: The Personalized `eToys` Home Page.

Consider the home page of the `eToys` web site shown in Figure 1. A simplistic ODMG schema of the Web site is as depicted in Figure 3 [1]. The database schema represents the a set of DTDs at the XML level and a **web-site-structure** for HTML pages. The structure of each web page in the given web-site, i.e., the **web-page-structure** at the HTML level and the DTD at the XML level is given by the definition of the respective class. Each object of a class corresponds to an XML document corresponding class-driven DTD. In our example, the home page for the `eToys` web site is modeled after the schema class `eToys`. Each element of a collection defined for the object is represented at the web-page level by a URL/URI link to the specific web-page of the corresponding object. Thus, for example, in the `eToys` class we have a collection of objects of the type `eToysByAge`. For the home page, this collection is represented by the links `0-12 months`, `2 year`, `4 year`, etc. Each of these URLs links to the web-pages `0-12 months`, `1 year`, etc., i.e., the representations of the objects of the class `eToysByAge`. Atomic literals such as the `String` attributes in a class are displayed as **web-items**. For example, text descriptors such as the `welcomeMessage` are shown on the **web-page** as plain text. In teh web generation process we first convert the OODB schema to a set of DTDs and corresponding XML pages and then apply standard technology such as XSL stylesheets to generate the HTML pages. A synopsis of the web-mapping from ODMG to XML and ODMG to HTML is given in Table 1.

SERF: Re-Structuring at the Database Level. The strength of our approach lies in the fact that at the database level, we can exploit existing services such as complex re-structuring to provide the same rich capabilities for the web pages themselves. A re-structuring or a view desired for the web-site is translated to an equivalent transformation template at the database level which then is responsible for the production of the schema. The new view schema is translated using the ODMG to XML mapping (Table 1) to an equivalent XML view.

We provide the SERF framework [RCC00] to enable *user-customized* and possibly *very complex* database schema transformations thereby supporting a *flexible*, *powerful* and *customizable* way of generating web-pages. A declarative approach to writing these transformations is to use OQL together with a view mechanism. OQL

[1] This schema has been fabricated for the purpose of our example here, and does not reflect the `eToys` schema.

ODMG Primitives	XML Concepts	Web Constructs
Schema	**set of DTDs**	**Web-site-structure**
Type	**DTD**	**Web-page-structure**
Object	**XML document**	**Web page**
OID	**URI**	**URL**
Atomic literal	**Leaf element**	**Web-item**
Struct literal	**Internal element**	**Structured web-item**
Collection of literals	**Collection of elements**	**List of web-items**
Extent of a type	**XML documents of a DTD**	**Web pages of a web-page-structure**

Table 1: Web Semantics Mapping between ODMG, XML and Web Semantics

can express a large class of view derivations and any arbitrary object manipulations to transform objects from one type to any other type.

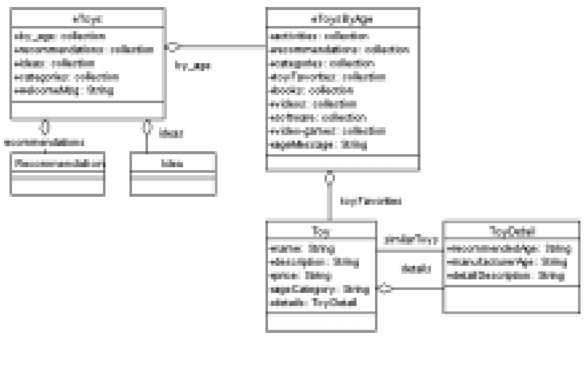


Figure 3: The ODMG Schema for the Web-Page in Figure 1.

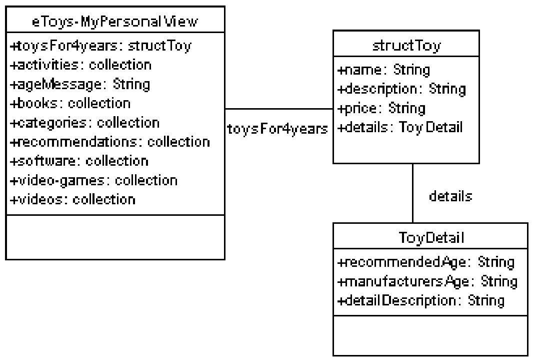


Figure 4: The ODMG Schema for the Web-Page in Figure 2.

Consider, for example that instead of the generic existing `eToys` home page, `eToys` desired a home page more geared towards the individual user, a user who wants to view only the recommended toys for `4 year` olds [2]. Moreover, consider that the user wants to see the brief description of all recommended toys on one web-page rather than having to click to get to the toys. Figure 2 shows a representation of the personalized web page. To accomplish this, we create a new view schema at the database level. Thus, instead of having the generic `eToys` schema, we have a personalized `PeToys` view schema. The `PeToys` schema contains information only on the recommended toys for `4 year` olds. Moreover, the `eToys-MyPersonalView` class contains an *inlined* description of all the toys along with additional pertinent information. The class `Toy` is stored in a structure. Figure 4 shows the new view schema while Figure 5 shows the OQL transformation used to accomplish the re-structuring, i.e., to produce the new view schema.

However, writing these transformations for the re-structuring of the database is not a trivial task as the view definition queries can be very complex. Similar to schema evolution transformations, it is possible to identify a core set of commonly applied transformations [RCC00]. For example, flattening a set of objects such that they appear as a list of web-items rather than a list of URLs is a common transformation that can be applied for different web site schemas. Thus in our framework we offer *re-use* of transformations by encapsulating and generalizing them and assigning a name and a set of parameters to them. From here on these are called *view transformation*

[2]The company would of course need to have a process in place that allows the user to specify a profile or use a filtering mechanism to generate the profile before they can build these personal web pages.

```
// define a named query for the view
define ViewDef (viewClass)
   select c
   from viewClass c
   where c.age-category = “4year”;

// create view eToys-MyPersonalView from the eToysByAge class
// and create struct structToy from the Toy class

create_view_class (eToysByAge, eToys-myPersonalView, ViewDef(eToysByAge ));

create_view_struct (Toy, structToy);

// add an attribute structToy to the view
add_attribute (eToys-MyPesonalView, structToy,
                collection<structToy>, null);

// get all the objects of class
define Extents (cName)
   select c
   from cName c;

// for each of the eToys-MyPersonalView object, find its referred Toy
// objects via the toyFavorites attribute and convert them into a
// a collection of  structToy.
define flattenedCollection (object)
   select (structToy)p.*
   from Toy p
   where exists (p in object.toyFavorites);

for all obj in Extents (eToys-MyPersonalView )
   obj.set(obj.structToy, flattenedCollection(obj));
```

Figure 5: SERF Inline Transformation.

```
begin template convert-to-literal (Class mainclassName,
                                   String mainViewName,
                                   Attribute attributeToFlatten,
                                   String structName)
{
   // find the class that needs to be flattened given the attribute name
   refClass = element (
                 select a.attrType
                 from MetaAttribute a
                 where a.attrName = $attributeToFlatten
                 and a.classDefinedIn = $mainclassName );

   // Create the view class
   create_view_class ($mainclassName, $mainViewName, View ($mainclassName));

   // flatten refClass to a struct
   create_view_struct ($refClass, $structName);

   // add a new attribute to hold the struct
   add_attribute ($mainViewName, $structName, collection<$structName>, null);

   // get all the objects of class
   define Extents (cName)
      select c
      from cName c;

   // convert a collection of objects to a collection of structures.
   define flattenedCollection (object)
      select ($structName)p.*
      from $refClass p
      where exists (p in object.$attributetoFlatten)

   for all obj in Extents ($mainViewName)
      obj.set(obj.$structName, flattenedCollection(obj));

   // remove the attributetoFlatten attribute
   delete_attribute ($mainViewName, $attributeToFlatten);
```

Figure 6: SERF Inline Template.

templates or *templates* for short. Figure 6 shows a templated version of the transformation in Figure 5. This template can be applied to *inline* the information linked in by a reference attribute (URL at the XML level) into the parent class (web-page) itself.

We have also proposed the development of a library of such templates. This is an open, extensible framework as developers can add new templates to their library, once they identify them as recurring. We envision that a template library could become an important resource in the web community much like the standard libraries in the programming environment.

Maintenance of Personal Views. Once the personal views are deployed, it is essential to have a maintenance process in place to provide an end-to-end solution. This is required to handle not only the data updates, for example a new toy suitable for `4 year olds` is added to the database (data update - base to view), but also to deal with changing tastes, such as the user is now interested in toys for a `2 year old` in addition to the toys for a `4 year` old (schema change - view to base), or to deal with a new feature such as a special 25% sale on popular items (schema change - base to view). All of this information needs to be propagated from the base schema to all of the view schemas in the system or conversely may need to be propagated from the view schema to the base schema. Moreover, with e-commerce there is an additional dimension of security. The personalized views may contain user-sensitive information and hence unlike the home page should not be visible to the world. Thus security permissions need to be attached to each personal view. This would often require propagation from the base to the view in the case where the base is the keeper of all security information, and from the view to the base in the case of the user wanting additional individuals to share the same web-page.

To allow for the diversity in the type of information and the direction of propagation, we have designed a rule-based propagation framework as opposed to constructing hard-coded algorithms to handle the same. Rules for our framework can be expressed using a rule-based extension of OQL, PR-OQL. These rules are specified for each object algebra node in the derivation tree. They also specify the direction in which the information is propagated, `up` from the base to the view or `down` from the view to the base. For example, a text message that announces to the customers a special 25% sale on the top ten toys can be added to the base class via a schema change `add-attribute(eToys, saleMsg, String, ‘‘Special sale on top ten toys’’)`. Using the rules in our framework, this change is propagated to all the defined views in an automated fashion. Figure 7 shows the syntax of the rules defined in our framework while Figure 8 shows a sample rule.

define [**propagation**/**re-write**] **rule** *ruleName* **for** *class* :
on *event*
in *direction*
when *condition(s)*
do [**instead**] *action(s)*
precedes [*rule*]$^+$

Figure 7: Syntax of a Rule.

define re-write rule *rule1* **for** *project* :
on *add-attribute(C,a,t,default)*
in *up*
when
do *re-write-query(existingQuery, a)* &&
propagate-to-derivedNodes
precedes *rule3*

Figure 8: A Sample Rule.

3 Summary

We have developed a prototype for the Re-Web system, a Java-based system (JDK1.1.8 and Java Swing). We make use of the LotusXSL and IBM XML parser for the generation of web pages from underlying databases. The system has been implemented and tested on WinNT and Solaris. Figure 9 gives the general architecture of Re-Web. The Re-Web system will be demonstrated at Sigmod 2000 [RCC00].

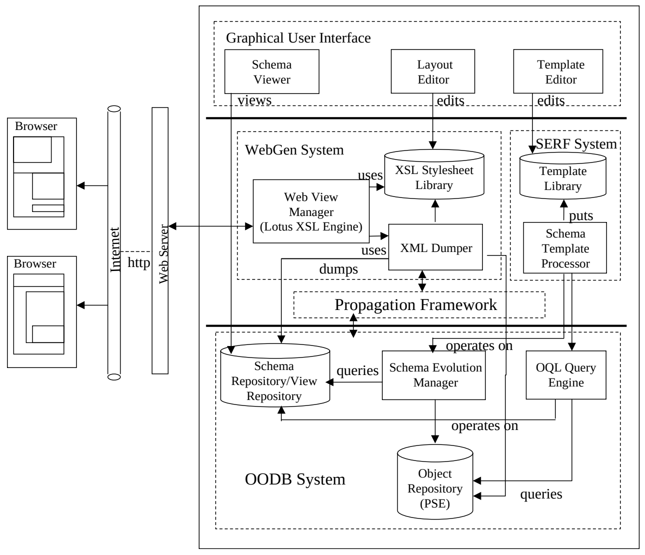


Figure 9: Architecture of the Re-Web System.

In summary, here we have given an overview of our Re-Web system and its prominent features such as:

- Generation of XML/HTML web-pages from an ODMG schema.
- Personal Views using re-structuring and view capabilities at the database level.
- Maintenance of Personal Views using an extensible rule-based propagation framework.

References

[AGM$^+$97] Serge Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for Semistructured Data. In *Workshop on Management of Semistructured Data*, pages 83–90, 1997.

[AMM98] P. Atzeni, G. Mecca, and P. Merialdo. Design and Maintenance of DataIntensive Web Sites. In *EDBT'98*, pages 436–450, 1998.

[AY00] C. C. Aggarwal and P.S. Yu. Data Mining Techniques for Personalization. In *IEEE Bulletin - Special Issue on Database Technology in E-Commerce*, in this issue.

[CR00] L. Chen and E. A. Rundensteiner. Aggregation Path Index for Incremental Web View Maintenance. In *The 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-based Information Systems, San Jose, to appear*, June 2000.

[CRCK98] K. Claypool, E.A. Rundensteiner, L. Chen, and B. Kothari. Re-usable ODMG-based Templates for Web View Generation and Restructuring. In *WIDM'98*, pages 314–321, 1998.

[FFLS97] M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for a Web-Site Management System. *SIGMOD*, 26(3):4–11, September 1997.

[RCC00] E.A. Rundensteiner, K.T. Claypool, and L. et. al Chen. SERFing the Web: A Comprehensive Approach for Web Site Management. In *Demo Session Proceedings of SIGMOD'00*, 2000.

An Interoperable Multimedia Catalog System for Electronic Commerce

M. Tamer Özsu and Paul Iglinski
University of Alberta
{ozsu,iglinski}@cs.ualberta.ca

Abstract

We describe our work in developing interoperable smart and virtual catalogs for electronic commerce which incorporate full multimedia capabilities. Smart catalogs enable querying of their content. The underlying assumption of our work is that future catalogs that will be used in electronic commerce will have two characteristics: (a) they will not only contain text and (perhaps) images, but will contain full multimedia capabilities including audio and video, and (b) these catalogs will not be monolithic, consisting of storage in a single database management system (DBMS), but will be distributed involving a number of different types of data storage systems. The general approach incorporates multimedia data management capabilities and interoperability techniques for the development of distributed information systems. This work is part of a larger project on electronic commerce infrastructure involving multiple universities and companies in Canada.

1 Introduction

Most of the existing electronic commerce (e-commerce) catalogs (both in business-to-business and in business-to-consumer e-commerce) are text-based with limited multimedia capabilities, only incorporating still images. This limits the "shopping experience" of customers. Catalogs should have full multimedia capabilities with the inclusion of advertisement or training videos and audio "talk-overs". This enables the incorporation of TV and radio commercials into the catalogs.

In this paper we discuss our current work in developing catalog servers with multimedia capability. One identifying characteristic of our work is that catalog servers are developed using database management system (DBMS) technology. The use of DBMS technology does not require justification for the readers of the *Data Engineering Bulletin*. However, suffice it to point out that the declarative query capabilities of DBMSs facilitate the development of smart catalogs that allow sophisticated querying of their content [KW97].

Existing catalog systems are generally proprietary, monolithic systems which store all data in one system. This is not a suitable architecture for the types of catalog systems that are envisioned in this project. The proposed catalog systems have to store and manage multiple types of media objects (text, images, audio, video, and synchronized text), as well as meta information about these objects. Moreover, these media objects have different characteristics that require specialized treatments. Therefore, in this project we propose an open architecture that uses different specialized servers for different classes of media objects and combines them to achieve a multimedia catalog server.

Another characteristic of catalog servers for Internet-based e-commerce is the significant distribution of the servers. Note that we make the distinction between "Internet-open" and "Internet-based". The former are systems which allow access, over the Internet, to catalogs. Most of the existing catalog servers are in this group and they are either centralized systems or distributed in a small scale. These types of systems would have difficulty in supporting sophisticated electronic commerce applications such as virtual malls. In this research we investigate solutions where data are distributed more widely and access is provided over the Internet. These are what we call Internet-based catalogs.

It is unlikely that all catalogs in an e-commerce environment will follow the same design or that all of the catalog information will be stored in one catalog server. The environment will include repositories other than DBMSs while still providing a DBMS interface and functionality. This raises issues of interoperability and the development of virtual catalogs that "dynamically retrieve information from multiple smart catalogs and present these product data in a unified manner with their own look and feel, not that of the smart catalogs" [KW97].

In Section 2 we provide an overview of the architectural approach that we have adopted in this research. Section 3 discusses the structured document database that is part of the catalog server while Section 4 is devoted to a discussion of the image DBMS. Section 5 presents highlights of on-going work.

2 Architecture

The fundamental architectural paradigm that we use in this research is one of loose integration between system components. This is reflected both in our development of the individual catalog servers and in the manner that we federate multiple catalog servers into one e-commerce environment.

A catalog server with full multimedia capabilities requires support for various media: text, still images, video, audio, VRML objects, etc. Even though there have been attempts to support all of these media types within a single multimedia DBMS, the end result is generally not satisfactory. The requirements of individual media are different, their requirements from the underlying storage systems vary and the types of queries that need to be supported are significantly different. "Monolithic multimedia DBMS" attempts typically treat media data as blobs without much support for content-based querying and access.

In our work, we take a different approach which is based on the development of individual DBMSs that support each media type. Thus, we have currently developed a text DBMS [ÖSEMV95, ÖSEMJ97] and an image DBMS [OÖL$^+$97] and are working on a continuous media DBMS to support audio and video. This raises the question of the combination of various media types in a complex multimedia object. The paradigm that we use for complex multimedia objects is that of a multimedia document. In other words, we consider that the individual media objects (e.g., a text description of a product, its image and a demonstration video clip of the product) to be organized into a multimedia document with a particular structure. Thus, it is possible to access individual objects directly as well as accessing them as parts of a larger unit. Figure 1 describes our overall architecture in a very abstract fashion.

As indicated above, catalog servers for Internet-based e-commerce applications are likely to be distributed and heterogeneous. Despite some claims that catalog information of multiple vendors will be integrated in one server, we consider this to be highly unlikely and unrealistic. What will typically happen is that each vendor will develop its own catalog and will bring it to a federation (e.g., in the form of a virtual mall). They will undoubtedly be willing to follow certain standards to facilitate access, but they are unlikely to entirely "merge" their catalogs into a centralized catalog. Thus, the e-commerce environment has to address the classical interoperability issues that are exacerbated by the dynamism of the environment. An interoperable environment further allows each vendor to participate in multiple e-commerce federations. Thus, part of our work has been focused on flexible interoperability environments for large scale distributed systems. This work was conducted within the context of the AURORA project [YÖL97, Yan00], but space limitations do not allow us to discuss this aspect of our work.

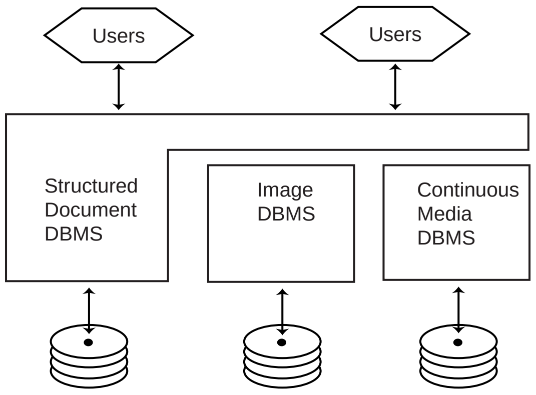


Figure 1: Catalog Server Architecture

3 Structured Document DBMS Component

As indicated above, the structured document DBMS [ÖSEMJ97] serves a dual purpose. It stores the catalog structure information, and thus has "links" to the other components, and it stores the text portion of the catalog. For the current prototype implementation, we have stored VRML objects as part of this DBMS as well, but the system stores VRML descriptions as code and does not provide facilities to exercise them.

The document structure is modeled using SGML. It is a client-server, object-oriented system that allows coupling with other servers (in particular continuous media servers) and handles dynamic creation of object types based on DTD elements.

The system is built as an extension layer on top of a generic object DBMS, ObjectStore [LLO$^+$91]. The extensions provided by the multimedia DBMS include specific support for multimedia information systems. Some of the unique features are the following:

1. It supports an extensible type system that provides the common multimedia types. The kernel type system is divided into two basic parts: atomic type system and element type system. The atomic type system consists of the types that are defined to model monomedia objects (i.e., objects that belong to a single media type). This part of the type system establishes the basic types that are used in multimedia applications. The element type system is a uniform representation of elements in a DTD and their hierarchical relationships. Each element defined in a DTD is represented by a concrete type in the element type system.

2. The system handles multiple DTDs and documents that conform to these DTDs within one database. The kernel type system extensibility is partly due to this requirement. This raises interesting type system issues. SGML, as a grammar, is fairly flat but allows free composition of elements. This, coupled with the requirement to handle multiple DTDs within the same database, suggests that the type system also be flat, consisting of collections of types (one collection for each DTD) unrelated by inheritance. This simplifies the dynamic type creation when a new DTD is inserted. However, this approach does not take full advantage of object-oriented modeling facilities, most importantly behavioral reuse. Instead of a flat type system, we implement a structured type system where some of the higher-level types are reused through inheritance. This has the advantage of directly mapping the logical document structure to the type system in an effective way. Furthermore, some of the common data definitions and behaviors for similar types can be reused at the discretion of the DTD developer. The disadvantage is that type creation is more difficult.

3. The system is able to analyze new DTDs and automatically generate the types that correspond to the elements they define. In addition, the DTD is stored as an object in the database so that users can run queries like "Find all DTDs in which a 'paragraph' element is defined." The components that have been imple-

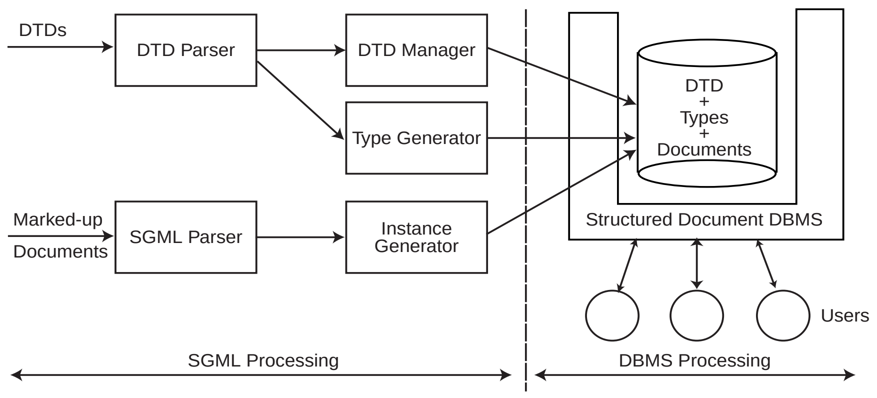


Figure 2: Structured Document DBMS Processing Environment

mented to support multiple DTDs are depicted in Figure 2. A DTD Parser parses each DTD according to the SGML grammar defined for DTDs. While parsing the DTD, a data structure is built consisting of nodes representing each valid SGML element defined in the DTD. Each DTD element node contains information about the element, such as its name, attribute list and content model. If the DTD is valid, a Type Generator is used to automatically generate C++ code that defines a new ObjectStore type for each element in the DTD. Additionally, code is generated to define a meta-type for each new element type. Moreover, initialization code is generated and executed to instantiate extents for the new element objects and to create single instances of each meta-type in the specified database. A Dtd object is also created in the database. This object contains the DTD name, a string representation of the DTD, and a list of the meta-type objects that can be used to create actual element instances when documents are inserted into the database.

4. The system automatically handles the insertion of marked-up documents into the database. Many systems have facilities for querying the database once the documents are inserted in it, but no tools exist to automatically insert documents. This is generally considered to be outside the scope of database work. We have developed tools to automate this process. The SGML Parser accepts an SGML document instance from an Authoring Tool, validates it, and forms a parse tree. The Instance Generator traverses the parse tree and instantiates the appropriate objects in the database corresponding to the elements in the document. These are persistent objects stored in the database that can be accessed using the query interface. The parser is based on a freeware application called *nsgmls*. The parser was modified and linked to DTD specific libraries to incorporate the necessary changes.

4 Image DBMS Component

The image objects of the catalog server can alternatively be stored in the document database without content annotations or in a separate image DBMS, called DISIMA [OÖL$^+$97, OÖIadC00], with enriched content model enhancements. The objective of DISIMA is to go beyond similarity-based querying and enable content-based querying. The system, whose architecture is depicted in Figure 3, stores images and meta-data in an object DBMS (ObjectStore) and provides a high level declarative query language. The catalog server utilizes DISIMA in managing the images within catalogs. The availability of a full-featured image DBMS as part of a catalog server enables users to perform catalog searches based on images in addition to text.

DISIMA is composed of the querying interfaces (MOQL and VisualMOQL), the meta-data manager, the image and salient object manager, the image and spatial index manager, and the object index manager. The interfaces provide several ways (visual and alpha-numeric) to define and query image data. The data definition language (DDL) used for the DISIMA project is C++ODL [CBB$^+$97] and the query language MOQL is an extension

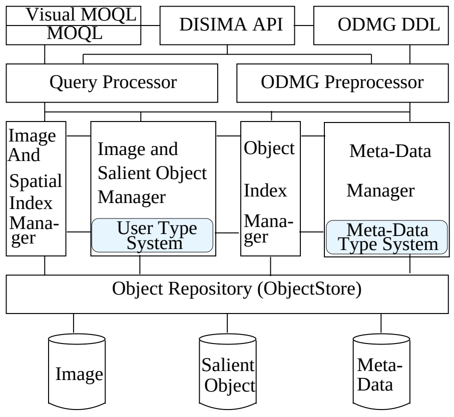


Figure 3: DISIMA Architecture

of OQL [LÖSO97]. The DISIMA API is a library of low-level functions that allows applications to access the system services. DISIMA is built on top of ObjectStore, which is used primarily as an object repository (since it does not support OQL). The image and salient object manager implements the type system, and the meta-data manager handles meta-information about images and salient objects [OÖL+97]. The index managers, under development, allow integration of user-defined indexes.

In DISIMA, an image is composed of salient objects (i.e., interesting regions). The DISIMA model is composed of two main blocks: the image block and the salient object block. A *block* is a group of semantically related entities. The *salient object* block is designed to handle salient object organization. For a given application, salient objects are identified and defined by the application developer. The definition of salient objects can lead to a type lattice. DISIMA distinguishes two kinds of salient objects: physical and logical salient objects. A *logical salient object (LSO)* is an abstraction of a salient object that is relevant to some application. A *physical salient object (PSO)* is a syntactic object in a particular image with its semantics given by a logical salient object. A PSO has a shape, which is a geometric object stored in its most specific class [OÖIL99]. The separation of the physical and logical salient objects allows users to query the database according to either the physical disposition of salient objects in an image or with regard to their existence.

The image block is made up of two layers: *image* layer and *image representation* layer. We distinguish an image from its representations to maintain an independence between them. At the *image* layer, the application developer defines an image type classification which allows categorization of images.

In addition to MOQL as a text-based query language, DISIMA provides a visual querying interface called VisualMOQL [OÖX+99]. MOQL extends the standard object query language OQL by adding spatial, temporal, and presentation properties for content-based image and video data retrieval as well as for queries on structured documents. VisualMOQL implements only the image part of MOQL for the DISIMA project. Although the user may have a clear idea of the kind of images he/she is interested in, the expression of the query directly in MOQL is not necessarily straightforward. VisualMOQL proposes an easier, visually intuitive way to express queries, and then translates them into MOQL.

5 Conclusions

We discussed our current work in developing catalog servers with multimedia capability. The catalog server uses specialized DBMSs for individual media types and integrates them into a single server. The current version accommodates text, images, and VRML objects and this version is operational. Continuous media capabilities are currently being added. In this paper we provided an overview of the existing component systems and the way they are integrated.

There is ongoing work along a number of fronts. First, as indicated above, is the incorporation of continuous media capabilities. The first prototype of this version will be operational in late spring of this year. The second line of work is the conversion of the structured document DBMS to support XML rather than SGML. With the integration of the various component systems, the issue of a unifying query interface becomes important. Our approach to this issue is to pursue extensions to the MOQL language that we have developed for images and video access. As part of this research, we have ongoing efforts on XML query languages and their optimization. We are also developing a distributed version of the DISIMA image DBMS over a CORBA infrastructure. The distributed version will be demonstrated at the upcoming SIGMOD Conference.

Acknowledgements

This research and the development of the structured document DBMS is supported by by a grant from the Canadian Institute for Telecommunications Research (CITR) under the NCE Program of the Government of Canada. The development of the DISIMA distributed image DBMS was supported by a strategic grant from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

[CBB+97] R.G.G. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade, editors. *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, 1997.

[KW97] R. Kalakota and A.B. Whinston, editors. *Readings in Electronic Commerce*, chapter 11, pages 259–274. Springer-Verlag, 1997.

[LLO+91] C. Lamb, G. Landis, J. Orenstein, , and D. Weinreb. The ObjectStore database system. *Communications of the ACM*, 34(10):50–63, October 1991.

[LÖSO97] J. Z. Li, M. T. Özsu, D. Szafron, and V. Oria. MOQL: A multimedia object query language. In *Proc. 3rd Int. Workshop on Multimedia Information Systems*, pages 19–28, September 1997.

[OÖIadC00] V. Oria, M.T. Özsu, P. Iglinski, and B. Xu an dE. Cheng. DISIMA: An object-oriented approach to developing an image database system (demo description). In *Proc. 16th Int. Conf. on Data Engineering*, March 2000.

[OÖIL99] V. Oria, M. T. Özsu, P.J. Iglinski, and Y. Leontiev. Modeling shapes in an image database system. In *Proc. 5th Int. Workshop on Multimedia Information System*, pages 34–40, October 1999.

[OÖL+97] V. Oria, M.T. Özsu, L. Liu, X. Li, J.Z. Li, Y. Niu, and P. Iglinski. Modeling images for content-based queries: The disima approach. In *Proc. 2nd Int. Conference on Visual Information Systems*, pages 339–346, December 1997.

[OÖX+99] V. Oria, M. T. Özsu, B. Xu, L. I. Cheng, and P.J. Iglinski. VisualMOQL: The DISIMA visual query language. In *Proc. 6th IEEE International Conference on Multimedia Computing and Systems*, Volume 1, pages 536–542, June 1999.

[ÖSEMJ97] M.T. Özsu, D. Szafron, G. El-Medani, and M. Junghanns. An object-oriented sgml/hytime compliant multimedia database management system. In *Proc. ACM International Conference on Multimedia Systems*, pages 239–249, November 1997.

[ÖSEMV95] M.T. Özsu, D. Szafron, G. El-Medani, and C. Vittal. An object-oriented multimedia database system for news-on-demand application. *ACM Multimedia Systems*, 3:182–203, 1995.

[Yan00] L.L. Yan. *Building Scalable and Flexible Mediation: The AURORA Approach*. PhD thesis, University of Alberta, Edmonton, Canada, 2000.

[YÖL97] L.L. Yan, M.T. Özsu, and L. Liu. Accessing heterogeneous data through homogenization and integration mediators. In *Proc. Second IFCIS Conference on Cooperative Information Systems*, pages 130–139, June 1997.

Database Design for Real-World E-Commerce Systems

Il-Yeol Song
College of Inf. Science and Technology
Drexel University
Philadelphia, PA 19104
song@drexel.edu

Kyu-Young Whang
Department of EE and CS
Korea Adv. Inst. of Science and Technology (KAIST) and
Adv. Information Technology Research Ctr (AITrc)
Taejeon, Korea
kywhang@mozart.kaist.ac.kr

Abstract

This paper discusses the structure and components of databases for real-world e-commerce systems. We first present an integrated 8-process value chain needed by the e-commerce system and its associated data in each stage of the value chain. We then discuss logical components of a typical e-commerce database system. Finally, we illustrate a detailed design of an e-commerce transaction processing system and comment on a few design considerations specific to e-commerce database systems, such as the primary key, foreign key, outer join, use of weak entity, and schema partition. Understanding the structure of e-commerce database systems will help database designers effectively develop and maintain e-commerce systems.

1 Introduction

In this paper, we present the structure and components of databases for real-world e-commerce systems. In general, an e-commerce system is built by following one of two approaches. The first approach is the customization approach using a suite of tools such as IBM's WebSphere Commerce Suite [Shur99]. For example, the Commerce Suite provides tools for creating the infrastructure of a virtual shopping mall, including catalog templates, registration, shopping cart, order and payment processing, and a generalized database. The second approach is the bottom-up development of a system in-house by experts of an individual company. In this case, the developer is manually building a virtual shopping mall with mix-and-match tools. In addition, a database supporting the business model of the e-commerce system must be manually developed.

Whether a developer is using the customization or the bottom-up approach, understanding the structure of e-commerce database systems will help the database designers effectively develop and maintain the system. Our paper is based on our experience of building real-world e-commerce database systems in several different domains, such as an online shopping mall, an online service delivery, and an engineering application.

The major issues of designing a database for e-commerce environments are [BD98, CFP99, KM00, LS98, SL99]:

- Handling of multimedia and semi-structured data;
- Translation of paper catalog into a standard unified format and cleansing the data;
- Supporting user interface at the database level (e.g., navigation, store layout, hyperlinks);
- Schema evolution (e.g., merging two catalogs, category of products, sold-out products, new products);
- Data evolution (e.g., changes in specification and description, naming, prices); Handling meta data;
- Capturing data for customization and personalization such as navigation data within the context.

In Section 2, we present our view of a value chain needed by the e-commerce system and its associated data in each stage of the value chain. In Section 3, we first discuss logical components of e-commerce database systems. We then present the detailed database schema of an e-commerce transaction processing (ECTP) system and discuss a few database design considerations specific to e-commerce systems. Section 4 concludes our paper with comments on the roles and future developments of e-commerce database systems.

2 An E-commerce Value Chain and Data Requirements

An e-commerce value chain represents a set of sequenced business processes that show interactions between online shoppers and e-commerce systems. A value chain helps us understand the business processes of e-commerce systems and helps identify data requirements for building operational database systems. Treese and Stewart [TS99] show a four-step value chain that consists of Attract, Interact, Act, and React. *Attract* gets and keeps customer interest. *Interact* turns interest into orders. *Act* manages orders; *React* services customers. The four-step chain could be considered as a minimal model for a working e-commerce system.

In this paper, we present a more detailed value chain that consists of eight business processes. The new value chain integrates steps such as personalization, which is usually performed by a separate add-on product. Figure 1 shows the integrated e-commerce value chain with the eight business processes, their goals and data requirements.

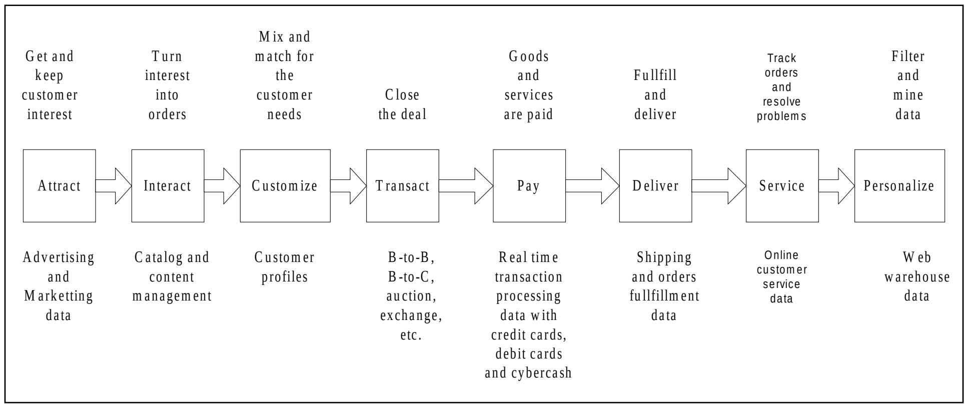

Get and keep customer interest
Turn interest into orders
Mix and match for the customer needs
Close the deal
Goods and services are paid
Fullfill and deliver
Track orders and resolve problems
Filter and mine data
Attract
Interact
Customize
Transact
Pay
Deliver
Service
Personalize
Advertising and Marketting data
Catalog and content management
Customer profiles
B-to-B, B-to-C, auction, exchange, etc.
Real time transaction processing data with credit cards, debit cards and cybercash
Shipping and orders fullfillment data
Online customer service data
Web warehouse data

Figure 1: An e-commerce value chain with eight business processes.

We call each phase of the value chain a business process in that it is important in its own right and involves significant complexity. Each business process involves a set of interactions between online shoppers and e-commerce systems for achieving particular objectives. Each business process will have different data requirements based on underlying business models supported by an e-commerce system. For example, products, services and users of e-commerce systems would be different whether the system supports B-to-B, B-to-C, auction, or exchanges.

A database designer must fully understand each business process and identify the data requirements needed to support each business process.

3 Database Schema for E-Commerce Systems

3.1 High-Level Logical Components

A database schema for a real-world e-commerce system is significantly complicated. Figure 2 shows a package diagram that shows logical components of a typical e-commerce database. The diagram uses the notation of Package used in UML [BRJ99]. A package in UML is a construct that groups inter-related modeling elements. In Figure 2, each package contains one or more related tables.

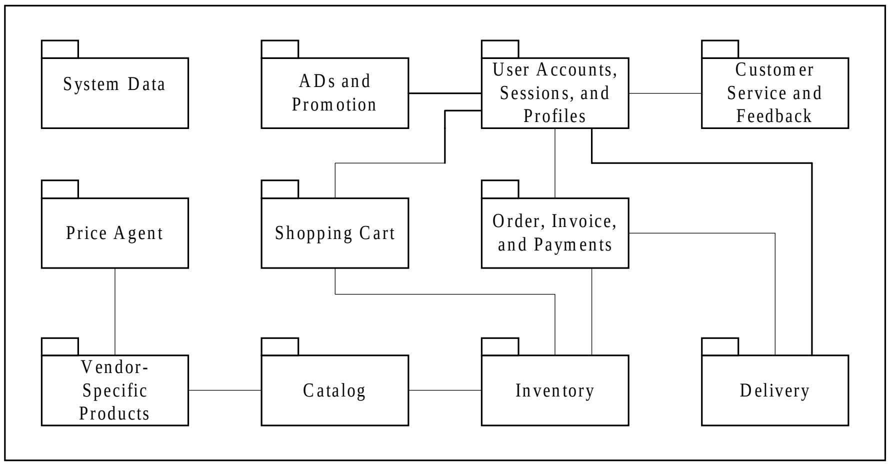


Figure 2: A Package diagram with logical components of e-commerce systems.

Package *User Accounts, Sessions, and Profiles (UASP)* records login ID, password, demographic data, credit card data, customer profiles, and usage history such as the total number of purchases, the total amount of payments, and the total number of returns. The package holds customization data, such as particular items to display, the amount of data to display, and the order of data presentation, and personalization data, such as user purchasing behavior, statistical data, and reporting data. The package could also be extended to include modules for capturing data for clickstream analysis [KM00]. The UASP package also keeps various user types such as individual customers, retail customers, user subgroups, buyers, sellers, and various affiliates, depending on the business model supported by the system.

Package *ADs and Promotion* involves tables that are related to advertising, promotion, and coupons. The package tracks which promotions are associated with which sessions, and which ADs are displayed in which sessions.

Package *Customer Service and Feedback* keeps tracks of customer feedback data for each user and order such as type, nature, status, and responses.

Package *Price Agent* contains data about competitions for each product type. The package allows the system to search other online websites that sell the same products and ranks the site names in the order of prices. Thus, the package may not exist if the site does not have any competitive sites in the same domain.

Package *Shopping Cart* models a shopping cart and inventory items in the shopping cart. Note that items in the shopping cart may or may not be actually ordered. Thus, the various states of the items must be kept track of.

Package *Order, Invoice, and Payment (OIP)* keeps track of actual orders that are transferred from a shopping cart. The package keeps track of individual order items, discounts, sale prices, commissions, taxes, cancels, and any changes in order states. The package also includes invoice history, payment history, and credit card processing modules.

Package *Delivery* includes shipping address, shipping methods and charges.

Packages *Inventory*, *Catalog*, and *Vendor-Specific Products* are closely related. *Vendor-Specific-Products* hold the vendor data and products. *Inventory* lists all the items that are available for sale and their associated tax data. *Catalog* is a standardized model that integrates various vendor-specific products in a domain. Since one vendor could use different terminology and format for the same product, *Catalog* protects the actual business model from the vendor-specific variations. *Catalog* plays a buffer between *Vendor-Specific Products* and *Inventory*. *Catalog* also needs a dynamic catalog management module to accommodate new vendors and new products that were not considered in advance. Thus, actual implementation of the program is done against *Catalog* and *Inventory*. Our experience shows that developing a proper schema for *Catalog* for a domain is most time-consuming and complex.

Package *System Data* holds various system parameters such as server configuration and access control parameters.

3.2 Schema for E-commerce Transaction Processing Systems (ECTP)

In this section, we present a database schema for an e-commerce transaction processing (ECTP) system that sells inventory items. Figure 3 shows the ECTP schema and includes some portions from UASP, Order-Invoice-Payment, Shopping Cart, Inventory, and Delivery packages. Even though a real-world schema would be much more complicated than the one shown in Figure 3, the schema illustrates a few interesting design considerations in e-commerce environments. The detailed schema will be different depending on the business rules and models supported by the site. Note that Figure 3 uses the UML notation. In order to create a relational schema from Figure 3, we need to add the primary key of the one-side to the many-side as a foreign key.

The proper selection of the primary key is very important. For most tables, we use a system-generated primary (surrogate) key to avoid dependence on data changes. Smart keys, which have embedded meanings, create dependency on those data. Thus, changes in those data cause changes in primary keys. All tables in Figure 3 uses surrogate keys.

Note that in order not to lose any data for the personalization procedure and report generation, there are cases that we do not declare an attribute, which is the primary key of another table, as a foreign key. For example, table ORDER_ITEM will capture attribute INV_ITEM# from INVENTORY_ITEM table in the relational schema. But we do not declare the attribute as a foreign key. Doing so will cause the referential integrity to be violated when the INV_ITEM# in the INVENTORY_ITEM table is deleted. When generating reports in this case, a join between ORDER_ITEM and INVENTORY_ITEM must use an outer join, rather than a natural join, so that the report include the order item with the deleted product.

In Figure 3, the relationship between *Order* and *OrderItem* is one-to-many. In typical data modeling situations, *OrderItem* is modeled as a weak entity of *Order*. Thus, the primary key of *OrderItem* will be a combination of Order# and a sequence number of items within the order number. However, in this diagram, we created Order_Item# as the primary key of *OrderItem*. Order# in *OrderItem* will simply be a foreign key, without being a part of the primary key of *OrderItem*. This allows the e-commerce system to easily and flexibly handle individual order items independently of *Order*. If you want to count the number of times an item was placed in order, even if the order itself was cancelled, this technique becomes useful.

On the other hand, *OrderItem History* was modeled as a weak entity of *OrderItem*. In *OrderItem History*, the history of each *OrderItem* can be recorded separately. For every traceable entities that could have changes in states, there must be a weak entity that keeps track of the history of the state changes. This facilitates the handling of various customer supports at the lowest level of grain. These traceable entities in Figure 3 include *User Account*, *Order*, *OrderItem*, *Invoice*, *Shipping*, and *Payment*. In Figure 3, we showed only *OrderItem History* and *Invoice History* as weak entities. *Invoice History* is a weak entity of *Invoice* and keeps track of all the changes in states of invoices.

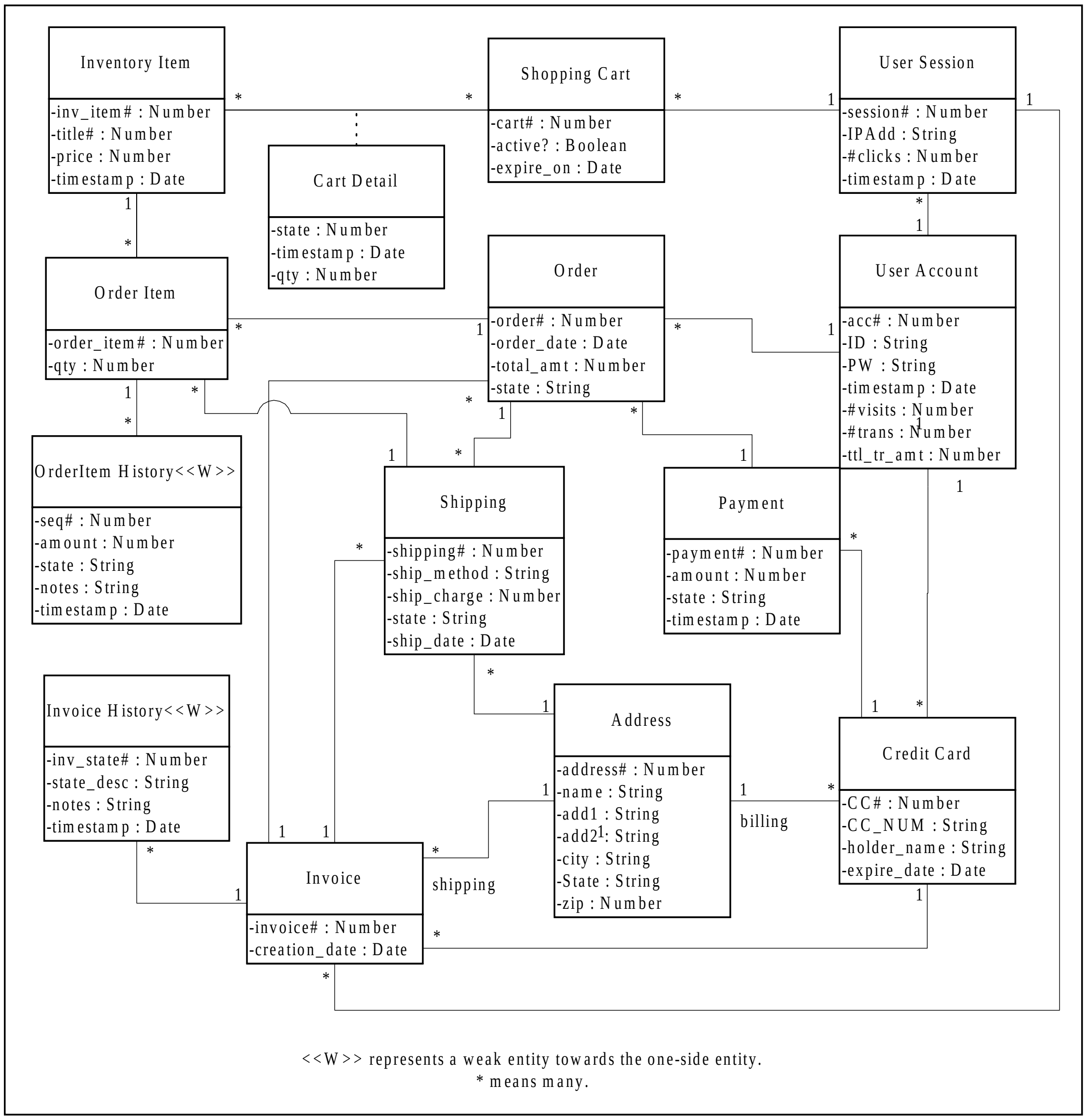


Figure 3: A simplified database schema for e-commerce transaction processing.

The relationship between *Invoice* and *User Session* allows the system to compute the purchasing rate, which is the number of sessions that result in an invoice, and eventually an actual order. The relationship between *OrderItem* and *Shipping* allows the system to send individual order item to different shipping address. *User Account* has a few statistical attributes. Usually they are placed in a separate table. In Figure, we showed them in *User Account* for the lack of space.

A typical real-world database schema for e-commerce systems are divided into multiple databases run by different database instances for several reasons. For example, a static schema such as catalog could be separated from a more volatile schema such as order processing. This multiple schema approach requires connecting those separated schemas when processing queries that need to concurrently access those schemas. Thus, this approach somewhat slows down the performance. However, there are many advantages such as better stability, maintainability, load balancing, and security.

4 Conclusion

In this paper, we presented an e-commerce value chain with eight business processes, logical components of e-commerce databases, and the schema for an e-commerce transaction processing system. Most real-world e-commerce database schema will have a similar framework as we presented in this paper. Our experience shows that e-commerce tools can speed up the development, but still lack certain functionality such as partial orders, back orders, returns, and email notification to users. Therefore, understanding the structure of e-commerce database systems will help the database designers effectively develop and maintain the system, regardless of the approach taken.

From the database design point of view, an interesting research issue is what database structures are needed to support customization and personalization most effectively. For example, how and what data do we need to capture to build a web warehouse [KM00] for personalization, and then, how do we communicate with users of systems? The personalization process requires significant resources to capture clickstream data and user behavior patterns. Readers are referred to the reference [SL99] for a list of typical OLAP queries and data warehouse design in e-commerce environments.

References

[BRJ99] Booch, G., Rumbaugh, J., and Jacobson, I., *UML User's Guide*, Addison Wesley, 1999.

[BD98] Buchner, A. and Mulvenna, M., "Discovering Internet Market Intelligence through Online Analytical Web Usage Mining," *SIGMOD Record*, Vol. 27, No.4, December 1998, pp. 54-61.

[CFP99] Ceri, S., Fraternalim P., and Paraboschi, S., "Design Principles for Data-Intensive Web Sites," *SIGMOD Record*, Vol. 28, No.1, March 1999, pp. 84-89.

[KM00] Kimball, R. and Merz, R., *The Data Webhouse Toolkit*, Wiley, 2000.

[LS98] Lohse, G.L. and Spiller, P., "Electronic Shopping," *Communications of the ACM*, Vol.41, No.7, 1998, pp. 81-88.

[Shur99] Shurety, S., *E-Business with Net Commerce*, Prentice Hall, 1999.

[SL99] Song, I-Y. And LeVan-Schultz, K., "Data Warehouse Design for E-Commerce Environments," *Lecture Notes in Computer Science*, Vol. 1727, Springer, pp. 374-388.

[TS99] Treese, G.W. and Stewart, L.C., *Designing Systems for Internet Commerce*, Addison Wesley, 1998.

End-to-end E-commerce Application Development Based on XML Tools

Weidong Kou
David Lauzon
William O'Farrell
IBM Toronto Lab, Canada

Teo Loo See
Daniel Wee
Daniel Tan
Nanyang Polytechnic, Singapore

Kelvin Cheung
Richard Gregory
Kostas Kontogiannis
University of Waterloo, Canada

John Mylopoulos
University of Toronto, Canada

Abstract

In this paper, we discuss an electronic business application framework and its related architecture. The framework is presented in the form of a prototype system which illustrates how XML tools can assist organizations on building and deploying e-commerce applications. The use of the system is presented in the form of a sample e-commerce application that involves shopping in a virtual store. The framework allows for customers and service providers to be linked through XML/EDI with back-end applications and to trigger e-procurement orders which are sent using either XML or EDI messages. Invoice handling and order tracking are also discussed along with extensions to the proposed architecture that allow for business process logic to be encoded at the server side, in the form of Event Condition Action scripts.

1 Introduction

E-business has become the focal point in the transformation of key business processes through the use of Internet technologies. By incorporating Internet technologies into core business processes, organizations are able to integrate back-end software applications and, data base systems into a seamlessly open environment [7], [6]. At the core of e-business lies electronic commerce or e-commerce, which is defined as commerce activities conducted electronically, particularly over the Internet, from online transactions to streamlined billing and payment systems.

Recent reports from investment and consulting firms (such as Goldman Sachs and First Data International) indicate that business-to-business (B2B) e-commerce related revenue over the next five years, can be as high as one trillion US dollars. E-commerce is with no doubt driving the fast growing digital economy in the new millennium.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

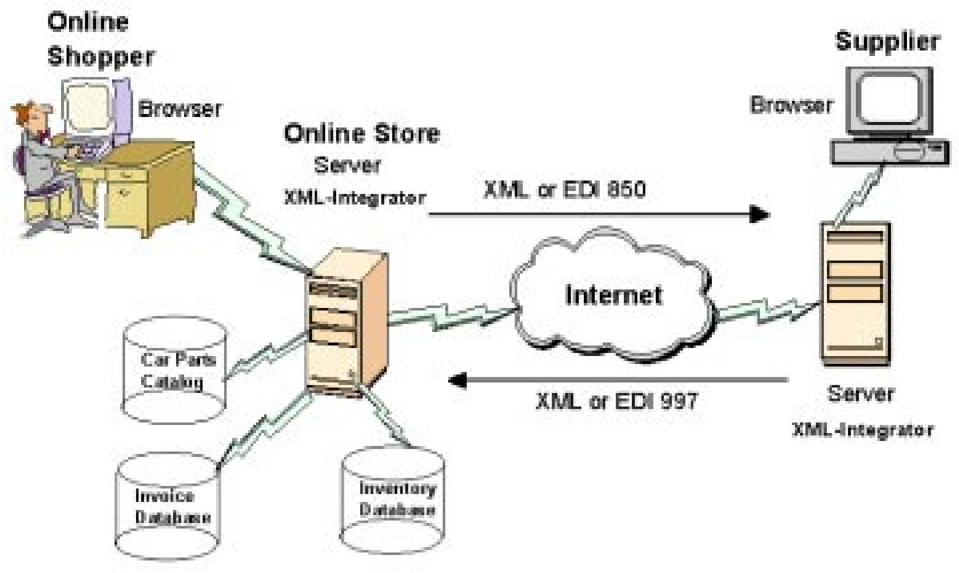


Figure 1: An e-Commerce Application Demo.

To meet the rapid growth of e-commerce, new technologies have emerged and new standards have been set. One such standard is XML (extensible markup language) [5] that provides an excellent vehicle to move data between applications and to allow for a smooth integration of new e-commerce applications from small and medium enterprises (SME), without requiring too much investment on complex or expensive EDI-based infrastructure.

Key e-commerce application areas such as inventory analysis, sales, accounting, and purchasing are only part of a larger framework that supports the selection, purchase, and delivery of a product to a customer. This framework can be modeled as a collection of information flows from one point to the next in the business process chain.

In this report, we present a prototype infrastructure that allows for the development of end-to-end e-commerce applications [2] using XML tools, and IBM's Websphere server environment. The motivation is to design and develop an architecture that allows different business partners to work together in a flexible way and, to deploy end-to-end e-commerce applications both in the business-to-business (B2B) and business-to-consumer (B2C) arena.

2 Prototype System Description and Discussion

A prototype implementation of an end-to-end e-commerce application was demonstrated at CASCON-1999, an annual conference co-sponsored by IBM and the Canadian National Research Council, in November 8-11, 1999 [2]. This prototype is based on a set of Internet technologies, including IBM XML tools, and Websphere, EJBs (enterprise Java Beans), and Java Servlets. The system aims to provide infrastructure for developing e-business applications by composing distributed web-enabled components.

The proposed environment focuses on typical e-commerce applications (Figure 1), in which multiple shoppers can browse a catalogue of an online store, select products, add them to a shopping cart, and place orders. The system provides integration with the underlying inventory system and, depending on the inventory information, a purchase order can be transmitted to a supplier in either XML or EDI (Electronic Data Interchange) 850

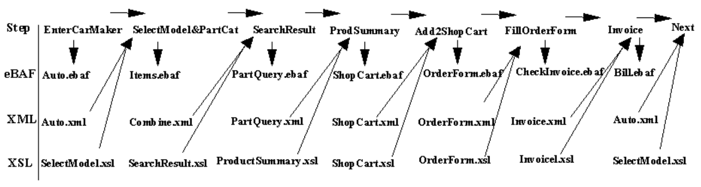


Figure 2: Flowchart of Shopping Steps And Associated XML, XSL, and eBaf Files.

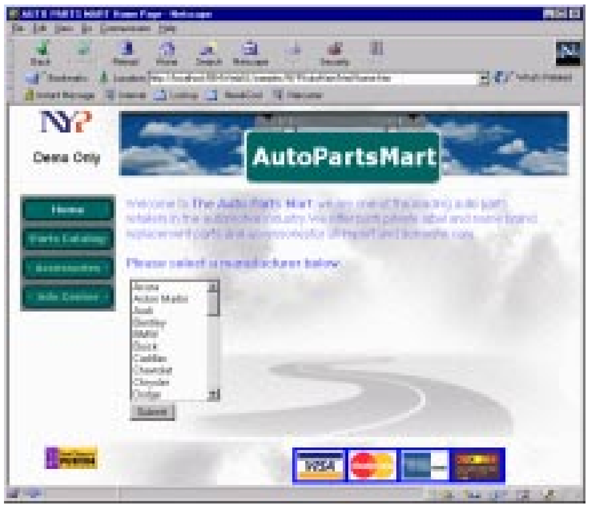


Figure 3: Home Page of the Car Parts Store.

format. The supplier responces can include acknowledgment messages in either XML or EDI 997 format. The online store can also track orders, bill the shoppers, and contact a transport company for delivery.

The prototype environment was built with IBM XML tools that are currently being developed at IBM [1] as part of the Electronic Business Application Framework (e-BAF). The e-BAF framework defines files which contain SQL scripts for an XML Integrator Servlet, data parameters provided by the user and, XSL style sheets for the presentation of results to the user in HTML.

The flowchart in Figure 2 llustrates each step that a shopper will go through with its associated XML, XSL, and e-BAF files.

1. Enter the AutoPartsMart Virtual Mall

   Figure 3 llustrates the home page of the demo for the Auto Parts Mart. It consists of an HTML page. The shopper selects the model of the car for which he would like a list of parts.

   Upon the selection of the manufacturer, this information is captured in an XML fragment of the form:

```
<?xmlversion="1.0"?><Input><XMLProject><Directory>\samples\NYPAutoMart
</Directory><EBAFile>Dauto</EBAFile></XMLProject><Fragment><Manufacturer>
Honda</Manufacturer></Fragment></Input>
```

   The fragment is sent to the XML Integrator Servlet running on the WebSphere Application Server. The fragment consists of the Project directory path for this demo, the e-BAF file which contains the script for the XML Integrator, and the data fragment that contains the parameters captured from the shopper.

   The XML Integrator Servlet parses the information in the XML fragment using the XML4J parser. The LotusXSL processor executes the script specified in the e-BAF file with the relevant data from the shopper that is captured in the data fragment. Queries to the database can be captured in the script and the result set is output in XML format by the XML Integrator Servlet. The final output from the XML Integrator is an HTML page that is composed from an XSL and an XML file. Refer to Figure 4 below.

   The DAuto.EBAF file is parsed by the XML Integrator Servlet. Data such as the selected car model is stored via the <SessionStore> tag where an SQL Query is made to the PARTS database to retrieve the model for the auto maker, the accessories sold by the Auto Mart, and the year of the model. The result of the query is captured in an XML file and the XML Integrator Servlet is then instructed to produce the final HTML file from this XML using the XSL file specified in the EBAF file. A sample of the HTML file is shown in Figure 5.

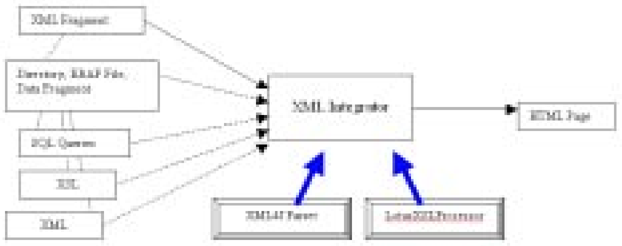


Figure 4: Input-output relationship of XML/XSL files and HTML page.

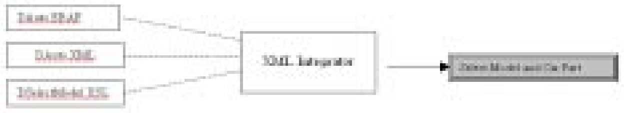
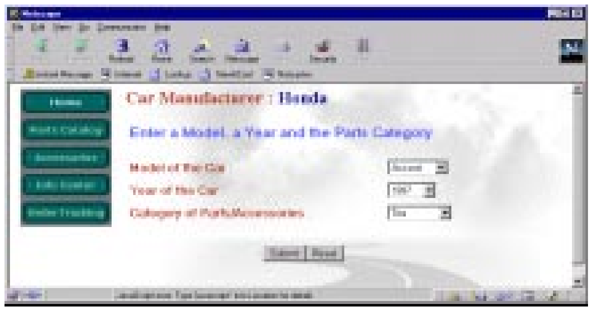


Figure 5: Logical relationship of files for ”Select Model and Car Part” step.

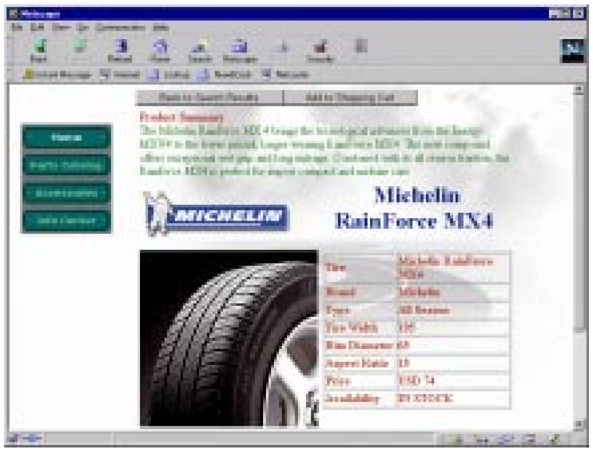


Car Manufacturer : Honda

Enter a Model, a Year and the Parts Category

Model of the Car	Accord
Year of the Car	1997
Category of Parts/Accessories	Tire

Figure 6: Screen of “Select Model and Car Part” Step.



Product Summary

[illegible]

Michelin RainForce MX4

Tire	Michelin RainForce MX4
Brand	Michelin
Type	All Season
Tire Width	195
Rim Diameter	65
Aspect Ratio	15
Price	USD 74
Availability	IN STOCK

Figure 7: Screen of “Product Summary” Step.

2. <u>Select Model and Car Part</u>

   In this step, illustrated in Figure 6, the shopper selects the model of his or her car and the part of interest, based on the car model data retrieved from the parts database. On clicking the “Submit” button, another XML fragment is sent to the XML Integrator. The flow (and subsequent flow) is similar to the previous step except that in the e-BAF file, the data fragments are now different. The processing is specified in the e-BAF file and in this case, it is Items.EBAF (see Figure 2). In this scenario, two queries have been initiated to the PARTS database in order to retrieve information as required by the Shopper. The results of the two queries are kept in two separate XML files and the XMLMerge tag is used to merge both files together. Finally, the information is presented to the shopper on an HTML page from this merged XML file using the specified XSL file.

3. <u>Search Result</u>

   The shopper chooses from the Search Result Page the part that is sought. The part number is captured and inserted into the fragment and sent to the XML Integrator Servlet. The PartQuery.EBAF file is used for specifying the queries to the PARTS database for retrieving the details regarding the selected parts. The results are merged together to form the Product Summary page (Figure 7).

4. <u>Product Summary</u>

   The shopper obtains as the result of the process the Product Summary page (Figure 7), which provides the details of the selected parts and accessories. On clicking the ”Add to Shopping Cart” button, another XML fragment is sent to the XML Integrator Servlet and the ”Shopping Cart” Page is displayed.

5. <u>Add to Shopping Cart</u>

   The shopper selects the number of tires that he wishes to purchase. The sub-total will be calculated for him. On

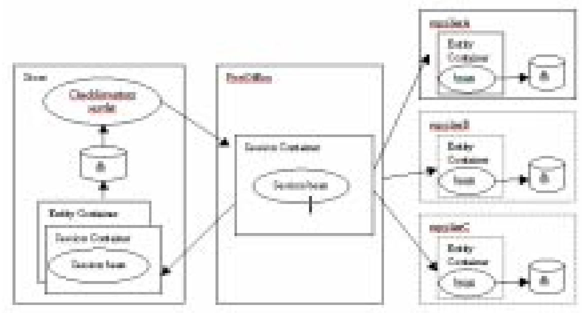


Figure 8: EJB for Information Transmission Between the Online Store and the Suppliers.

Figure 9: X12 850 Message to Supplier.

selecting Check Out, the XML fragment is sent to the XML Integrator Servlet. The Order Form is displayed to the shopper as shown in the next section with the session data pertaining to his purchase.

6. Fill in Order Form

   The Order Form shows the order details that has been provided by the shopper. The form is validated before the order can be placed. On clicking the ”Place the Order”, the XML fragment is sent to the XML Integrator Servlet, with the CheckInventory.EBAF in the EBAFile tag.

   In the CheckInventory.EBAF, the script uses the <Service> tag to invoke a service, CheckInventory, to perform replenishment of the PARTS database when the quantity-on-hand falls below the reorder level of the product after the purchase. For the demo, the CheckInventory service is implemented as a servlet that loads at startup and it registers itself with the XML Integrator's Service Handler. The CheckInventory Servlet invokes an Enterprise Java Bean (EJB) on the supplier server of the product when replenishment for the product is required. EJBs are used in this system to demonstrate how to simplify the process of developing the enterprise-class application systems. With EJBs, one could concentrate on writing the business logic rather than the complex middle-ware interfaces. Figure 8 illustrates its usage.

   The CheckInventory servlet reads from the database to determine the supplier address. The servlet then instantiates an EJBComms object (a simple utility class provided for this prototype) and provides it with the supplier node name and the filename that contains the X12 850 (Purchase Order) stream of data. The transmission is handled by calling methods of this class, namely, getConnection, activateBean and exitConnection. These methods establish connection with the postoffice (this can be the supplier itself) and remotely run a method of a session EJB, thereby transmitting the purchase order information. For persistence, the respective supplier Entity bean is being activated. The post office session bean will also trigger another connection with the online store to pass back the Functional Acknowledgment message X12 997.

   For the purpose of this prototype, the session bean will provide an asynchronous task to display the messages when they are received at the respective location. These are shown in Figures 9 and 10, respectively. As this is only a prototype, no business logic has been developed at the supplier end. The stream of data is not parsed at all and it is stored directly into the database.

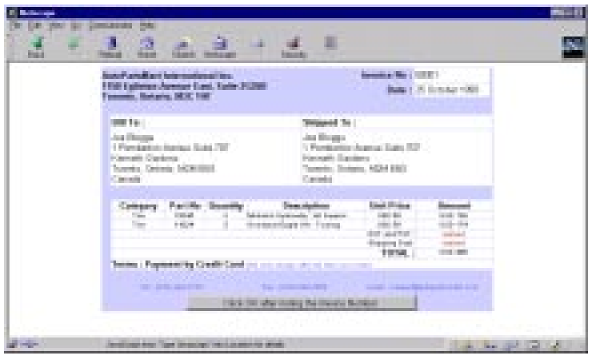
   The invoice is prepared for the shopper and displayed as shown in Figure 11.

7. Invoice

   The shopper takes note of the Invoice Number for the purpose of order tracking at a later stage. On clicking the “Click OK' button (see Figure 11), the XML fragment is sent to the XML Integrator Servlet and the shopper is shown the HTML page to select the model and car part at the Auto Parts Mart, as a means of indicating to the shopper that he or she can make another search for another category of part if he or she wishes.

[illegible]

Figure 10: X12 997 Message to Buyer.



[illegible]

Figure 11: Screen of "Invoice" step.

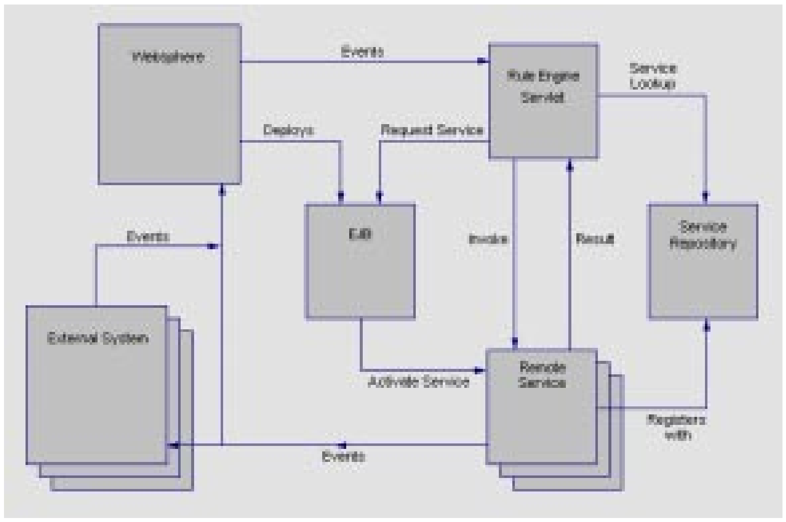


Figure 12: System Architecture.

3 Future Direction

The prototype presented in the previous section is based on pre-defined e-BAF files and on the XML Integrator. Such an architecture does not provide the kind of flexibility and extensibility which one expects in future e-commerce environments.

Figure 12 illustrates a generic architecture in which the XML Integrator presented in Figure 4 is replaced with a script language and a task enactment rule-based inference engine. Hard-coded static invocations to the various services employed in the prototype system are replaced with a script whereby business logic is encoded by a set of event-condition-action rules. The task enactment engine allows for rules to be triggered by accepting incoming events, which in turn may cause any number of rules to be triggered. For each rule triggered, a condition clause is checked and, if it is satisfied, the respective actions are invoked. In this way business process logic can be customized in the e-commerce environment to accommodate particular transaction scenaria or particular customer requirements.

In such an environment, control Integration deals with the composition, localization and invocation of remote services and components. Control Integration typically involves invoking a remote service. A service is defined informally as a method or program, that is, something that can execute either locally or remotely and may return a value. An essential point is that services can exist anywhere on a network or the Internet. Moreover, they execute at the system on which they reside or on a thin-client and they may be implemented in any language and run on any platform. This is in contrast to other paradigms whereby code is down-loaded and executed on the client side. Events are implemented as HTTP requests that encode the details of the events in XML format. Each rule contains components that have several typed parameters along with any other information pertaining to the events upon which the rules rely. This approach allows for a Web server to be used for capturing the events across the network and processing these events by a servlet. Another advantage with this architecture is that it allows for integration with other systems by translating events into a form other environments can use.

```
<ECAScript name="B2B Example">
  <ECARule name="Parts List Query">
    <Events> <EventExpr>
        <Event name="NeedPartList">
          <Param type="ModelName" name="Model" value=""></Param>
             <!--Note that "value" will be given a value (an-->
             <!--SQL query represented in XML at runtime    -->
        </Event>
    </EventExpr> </Events>
    <Conditions> </Conditions>
    <Actions>
      <Action name="Database Query">
        <Param name="Model"></Param> </Action>
             <!--The value given above will be used here-->
    </Actions>
  </ECARule>
  <ECARule name="Part out of Stock">
    <Events> <EventExpr>
        <Event name="NotInStock">
          <Param type="PartId" name="Part"></Param> </Event>
      </EventExpr> </Events>
    <Conditions> </Conditions>
    <Actions>
      <Action name="PlaceOrder">
        <Param name="Part" value = ""></Param>
        <Param name="Supplier" value="Toronto Car Parts"></Param>
            <!--The first value will be bound at runtime,-->
            <!--the second, when the script is read      -->
      </Action> </Actions>
  </ECARule>
</ECAScript>
```

Figure 13: Example ECA Rules in XML.

The scripting language allows tasks be specified as actions in a rule-based environment that implements the Event-Condition-Action (ECA) paradigm [4]. The scripts compose remote services in a way that customizes the business transaction logic in a B2B e-commerce application. XML is used for encoding the ECA scripting language, an example of which is provided in Figure 13. The service registration system is used for determining how the actions of this script are invoked. Finally, the rule engine is based on a forward-chaining activation mechanism that binds rule premises (Events), evaluates conditions and, triggers actions that in their turn produce new events.

4 Conclusion

In this paper, we first described a prototype system that facilitates e-commerce activities and is based on a set of Internet technologies, including XML, Java Servlets, EJBs, and IBM Websphere. Currently, we extend the prototype architecture to include ECA scripts so that, e-commerce applications can be developed and deployed in a customizable and extensible way. This leads to further development of a generic architecture using a Rule Engine to replace the XML Integrator. This new architecture will allow the user to customize e-commerce applications by encoding as ECA scripts specific business transaction logic.

References

[1] IBM XML Tools, http://www.alphaworks.ibm.com/ and http://www.ibm.com/developer/xml/

[2] W.Kou, et al, Demo: E-Commerce Application of XML Application Framework for e-Business, CASCON 1999, November 8-11, 1999.

[3] W.Kou and Y.Yesha, *E_Commerce Technology trends: Challenges and Opportunities*, Midrange Computing, Spring 2000, ISBN 158347-009-3.

[4] Mylopoulos, J., Gal, A., Kontogiannis, K., Stanley, M., ”A Generic Integration Architecture for Cooperative Information Systems” *in Proceedings of Co-operative Information Systems’96*, Brussels, Belgium, pp.208-217.

[5] XML Forum http://www.xml.org

[6] H.Ryan et. al eds. *Practical Guide to Client Server Computing* CRC Press, USA 1998.

[7] M. Brodie *Migrating Legacy Systems*, Morgan Kaufman Publishers, 1995.

Declarative Specification of Electronic Commerce Applications[*]

Serge Abiteboul, Sophie Cluet, Laurent Mignet
I.N.R.I.A.-Rocquencourt, France
Email: <firstname>.<lastname>@inria.fr

Tova Milo
U. Tel Aviv, Israel
Email: milo@math.tau.ac.il

Abstract

"Combining the existing semantics of EDI within an XML framework also makes possible large-scale automation of electronic commerce." *WebWeek* `XML/EDI Home Page`

In this short paper, we consider the use of a declarative language *(the* ActiveView language*) for specifying electronic commerce applications and, in general, a wide range of distributed applications based on sharing data and exchanging messages. The fact that the language is declarative allows the fast design, development and deployment and the flexible maintenance of such applications. The prototype* ActiveView *system that we implemented acts as an application generator that produces customizable Web applications with persistent data and basic workflow management.*

1 Introduction

The development of Electronic Commerce applications has benefited from several standards: (i) The **W**orld **W**ide **W**eb as a support for data exchange; (ii) e**X**tended **M**arkup **L**anguage [7] as a universal means of describing data; (iii) **E**lectronic **D**ata **I**nterchange (E.D.I [5]) as a common dictionary for EC data. However, the design, development, deployment and maintenance of Electronic Commerce applications are still very costly, tedious and time-consuming tasks even if application generators exist. This is in contrast with an economic context that pushes for quick response time to market demands.

A similar situation in the seventies led the database community to a declarative query language to access data, namely SQL. We believe that, similarly for EC applications, the solution to many problems passes via the use of a declarative query language. Such a language is presented here. The modeling of the data is based on XML and a query language for XML. We use here [4] but plan to adopt the standard query language for XML when available. The ActiveView language focuses primarily on what we believe is critical in these applications: the specification of (i) modes of sharing data and (ii) of communications between the various actors.

To validate the motivating ideas and the language, we implemented the ActiveView system and tested some EC application. The prototype is based on a three-tier architecture :

- The first tier represents the repository that stores XML data. (XML provides the desired flexibility in terms of semistructured data modeling [3].)

[*]The present work was supported by the French Ministry of Research within the framework of the R.N.R.T. GAEL project, a joint project between MatchVision, the L.R.I (Universit de Paris Sud) and INRIA.

- The second corresponds to a Java server application;
- The third consists of the end-user interface, based on standard Internet browsers.

A more detailed description of the ActiveView system can be found in [2]. The example used in this paper are based on a Web catalog application that we demonstrated at VLDB.

The paper is organized as follows. Section 2 introduces the ActiveView applications. Section 3 presents briefly the language. The last section is a conclusion.

2 Active Views

An ActiveView application allows different users to work interactively on the same data in order to perform a particular set of controlled activities. Before getting into details, let us illustrate the need to support such applications by considering an example. An electronic commerce application, say, a virtual store, typically involves several types of *actors*, e.g, customers and vendors. It also involves a significant amount of *data*, e.g. the products catalog (typically searched by customers) or the products promotion information (typically viewed by customers and updated by vendors). Observe that each of the actors may *view* different parts of the data (e.g. a customer can only see his/her own orders and the promotions relevant to his/her category, while vendors may view all the orders and promotions), each may perform different *actions* on the data, and have different *access rights*. Also, the requirements for *freshness* of data may differ. E.g. when a new promotion is entered, we may want to immediately refresh the customer screen so that the new promotion becomes visible. On the other hand, we may decide not to propagate all catalog changes to the user so as not to overwhelm communications with too many messages. When/if an order is issued, the user has to be notified if the price of some data actually ordered has been modified.

Each actor in an EC application typically performs several *activities*. E.g. a customer may be *searching* the catalog, *ordering* products, *changing* a passed order. Observe that in each of these activities, we may expect to show a different Web page to the actor that possibly includes only part of the available data and actions for that given actor. Observe also that actions performed by an actor may initiate other actions. For instance, when a customer orders a product, we may want to update the stock. Finally, note that it may be interesting to log some of the actors operations, providing a *trace* for later analysis or to settle possible disputes.

Functionalities The definition of an ActiveView application consists of:

1. the specification of the persistent application data (some XML document such as the product catalog). This is often provided by already existing applications;
2. the specification of the application in the active view language, namely AVL. Its compilation generates the run-time of the application and some default user interfaces.
3. the customization of the default user-interfaces.

Furthermore, it is possible to modify dynamically the application by adding and removing *active rules* (see Section 3).

As mentioned above, an ActiveView specification is a declarative description of an application. It specifies, for each kind of actor participating in the application: (i) the available data and operations for the given actor, (ii) the various activities, and (iii) some active rules. Thus, the general specification of an application is as shown in Figure 1.

ActiveView application application_name

ActiveView actor-kind$_1$ **in application** application_name
    view data
    methods
    activities
    active rules **...**
ActiveView actor-kind$_n$ **in application** application_name **...**

Figure 1: AVL general specification

Architecture The ActiveView system uses Axielle, the ArdentSoftware XML repository that provides standard database support. An ActiveView application consists of several independent clients of the repository communicating between them and with the repository through notifications. These clients are programmed in Java, and communicate with the server using the DOM interface [6]. Figure 2 shows the various components of a running application. As can be observed on the figure, the *active view application manager*, controls the activities of the (possibly many) *active view* clients. Active views receive/send remote method invocations from/to the end-users *interfaces* which runs on standard Web browsers.

In principle, the server, clients and interfaces may run on different machines. Typically, the interface are on remote systems. The view data is obtained from the repository by check-in/check-out. Repository changes are not in general immediately propagated to the active view. Mechanisms to allow immediate propagation if needed are provided. The active view and the interface see the same data.

The ActiveView application manager consists of the following modules:

(i) The update manager receives notifications of changes from the repository. It notifies the appropriate views of changes they are concerned with. This mechanism is based on a notification mechanism provided by Axielle.

(ii) The active rule module manages a set of rules specified by the application programmer. These rules are triggered by events generated by the application such as method firings or repository updates. The rule mechanism uses the JSDT Protocol developed by Sun Microsystems.

(iii) The tracing module keeps a log of specified events.

Together these modules provide the business intelligence of the application.

An active view is basically an object of our application. In the current version of the system, it is implemented in Java. An active view is generally related to an actual Web window opened by a user of the system. Some views independent of any interface may also be introduced, e.g., for bookkeeping. An active view has access to the repository as well as to some local data (the instance variables of the view object). It reacts to user commands and may be refreshed according to notifications sent by the application manager. The methods available on a view depend on the users access rights and may allow him/her to read, load, write, etc. part or the whole the data it sees.

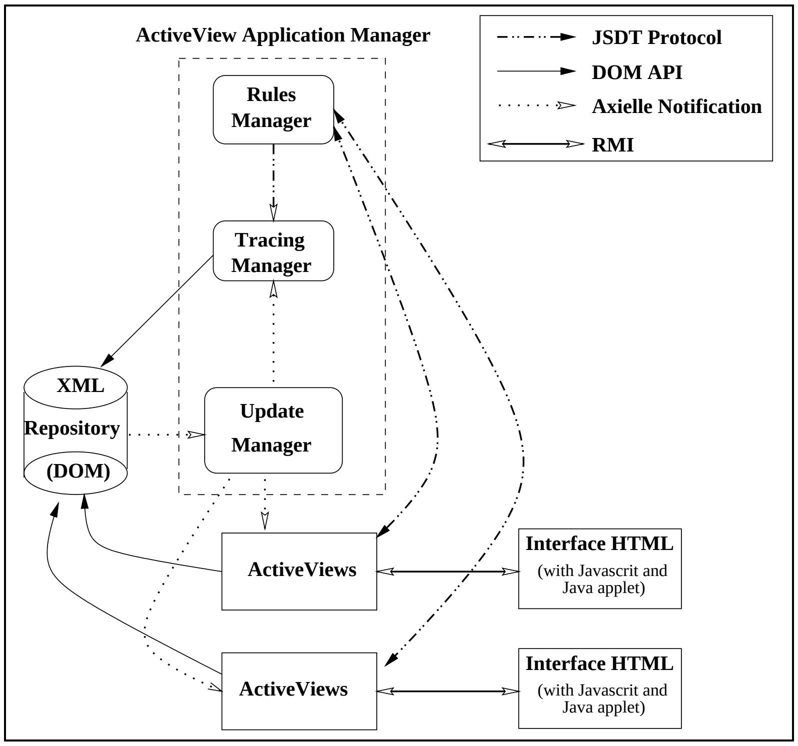


Figure 2: Architecture of an application

3 Activeview Language and User Interface

We describe briefly the language using a simplistic electronic commerce example in which we consider two kinds of users, namely, *Customers* and *Vendors*[1]. A vendor is mainly in charge of some customers and may interact with them, e.g., by offering them new promotions. More details on this application may be found in [2].

Data Specification A view may have access to persistent data (that exist independently of the particular actor) and transient data (that exist only during the actor's lifetime). The persistent data of a view are defined using XML-Queries [4] over the repository. Sophisticate access modes are provided. Consider, for instance, the *Customer's* and *Vendor's* views of the catalog defined as follows:

let	catalog : (CAT)$\star$
be	{select $m.CATS.CAT
	from $m in $catalog.META_CATS.META
	where $m.NAME == 'Musical Instrument' }
with	**self.**$\star$ X, X.ARTICLE Y, X.NAME Z
mode	**deferred read** X, **immediate read** Y Z

Customer's catalog definition

let	catalog : (CAT)$\star$
be	{select $m.CATS.CAT
	from $m in $catalog.META_CATS.META
	where $m.NAME == 'Musical Instrument'}
with	**catalog.**$\star$ X, X.ARTICLE Y, X.NAME Z,
	Y.PRICE P
mode	**read** X, **immediate read** Y Z, **write** P

Vendor's catalog definition

where *CAT* stands for category and *CATS* for categories. Customers and vendors see the same data except that a vendor may change the price of a product. The keyword *let* introduces some persistent data. (The keyword *local* for transient data is not used here.) Observe in the example how access modes are specified based on variables with bindings provided by a query.

[1]This example is based on the ActiveView Demonstration on VLDB

The *with* keyword permits to specify XML subtrees that are accessible in the view. The possible access modes are *write*, *append*, *remove*, *immediate read*, or *deferred read*. They allow to specify whether (some) updates are allowed and whether when an object is encountered, it should be loaded immediately (i.e., should we load the subtree rooted at this DOM node) or not.

The monitoring of persistent data changes is supported via two modes. The modes specify which actions should be taken when data changes are detected. If the keywords *let monitored* are used in place of simply *let*, the view is notified immediately of changes to that particular data. If *let fresh* are used, data changes are immediately propagated to the view.

Rules An actor specification may contain active rules. Rules can also be added and removed dynamically to modify the behavior of an application, e.g., to introduce a new kind of promotion. Rules [8] are very simple expressions of the form:

on (**local**)? <event> (**if** {<condition>})? **do** {<action>}

The events are (remote) methods calls (e.g., switch of activity), operations on instance variables or objects (i.e., write, read, append, remove) and change detections. The conditions are boolean XML queries. The actions are (remote) methods calls, operations on instance variables or objects, notifications or traces. A rule that is specified as *local* concerns this particular view and no event from or to other views.

As an example, we can introduce a new rule so that a vendor always sees what his/her customer is currently viewing:

```
on   display(element)
if   {sender.name == MyCustomer}
do   {self.display(element)}
```

Methods and Activities Due to space limitations, we will not describe here methods and activities specifications. In short, methods are defined in Java and are used for instance to do computations (AVL together with the XML query language should limit the need of Java code to the minimum). Activities are defined by simply listing the required data and methods (e.g., in the *browse* activity, we need the *catalog* variable and the *search* and *order* methods).

Users interfaces are currently based on dynamic HTML documents with embedded JavaScript and a Java applet to communicate with the active view. We plan to switch to XML as soon as XML browsers support the needed dynamic features. As for now, we generate XML "prescriptions", i.e., XML documents with application specific attributes and elements that describe the dynamic features of the document. These prescriptions are then compiled to dynamic HTML. There is one HTML document per user and per activity of that user. The applet is built on top of a generic API. In particular, the applet is application independent. The system generates default interfaces. The application programmer may redefine/customize either the XML prescription files or their corresponding HTML pages and JavaScripts.

4 Conclusion

The language AVL that is sketched here is supported by the ActiveView system [2]. There are limitations to the current architecture and prototype:

1. The combined use of JAVA RMI and Web Proxy limits the Web availability of ActiveView applications;
2. the use of one ActiveView Java client per user brings a lot of flexibility but would not scale to thousands of users;

3. the current ActiveView notification mechanism is rather complex. It uses both a database-centered notification mechanism from Axielle and JSDT. Although it supports nicely the toy application we developed, we believe it would not scale to thousands of notifications per seconds.

Acknowledgements A number of people helped implement the ActiveView prototype: Bernd Amann, Sbastien Ailleret, Frederic Hubert, Amlie Marian, Bruno Tessier, Brendan Hills. We thank Victor Vianu, Brad Fordham, Yelena Yesha for works with one the author [1] that initiated the present work. Finaly valuable comments were provided by Sihem Amer-Yahia, Anat Eyal, Jean-Claude Mamou, Jrme Simon and Anne-Marie Vercoustre.

References

[1] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 179–187, New York, USA, 1998.

[2] Serge Abiteboul, Bernd Amann, Sophie Cluet, Anat Eyal, Laurent Mignet, and Tova Milo. Active views for electronic commerce. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 138–149. Morgan Kaufmann, 1999.

[3] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publisher, October 1999.

[4] Vincent Aguilera. X-OQL. Technical report, INRIA, Verso Project, 2000. http://www-rocq.inria.fr/˜aguilera/xoql.html.

[5] Secretariat for federal edi. http://snad.ncsl.nist.gov/dartg/edi/.

[6] W3C. Document object model (DOM). http://www.w3.org/DOM.

[7] W3C. Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml.

[8] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan-Kaufmann, San Francisco, California, 1995.

TPC-W E-Commerce Benchmark Using Javlin/ObjectStore

Manish Gupta
Excelon Corporation
25 Mall Road, Burlingon, MA 01803
manish@exceloncorp.com

Abstract

Considerations during design for high performance and scalability of a typical high load e-commerce web system using an object database (ObjectStore), a middle tier transaction routing and scheduling library (Javlin) and standard commercial web servers are described. The TPC Benchmark(TM)- W specifies a set of operations to exercise a workload that includes simultaneous execution of a variety of transaction types, transaction integrity, secure online operations and dynamic page generation. We also present a subset of preliminary results from a particular implementation to illustrate tuning techniques that are representative of ObjectStore/Java based implementations.

1 Introduction

As the IDC predicts a figure of up to $1.3 trillion in internet commerce by the year 2003, more businesses are expecting a new generation of e-Commerce software implementations comprising of reusable modules and templates rather than custom development in order to reduce total software development costs. The average development cost of building an e-Commerce web site has dropped from the $1-$2 million range to the less than $200K range ([1]).With the explosion such e-Commerce software providers, businesses are naturally asking what implementations are best suited to their needs and system requirements. It is for this reason that the TPC (Transaction Processing Council) came up with their new TPC-W benchmark specification ([2]).

The TPC is a non-profit organization that works toward specifications of benchmarks to to serve as a resource of performance data for database and transaction processing systems. The TPC-W benchmark in particular measures functionality, performance and robustness of systems designed for transactional web environments including business-to-consumer, business-to-business and intranet.

At Excelon Corporation, we have implemented this pecification using a suite of products, including an object database (ObjectStore) and a middle-tier cache distribution library (Javlin). In this paper we first briefly discuss why the chosen product line is a good fit for the e-Commerce problem domain and what our goals from the implementation were. We then outline how our system was designed to satisfy requirements of the specification. Finally we present some preliminary results to illustrate the process of performance tuning.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

2 e-Commerce, ObjectStore and Javlin

The e-Commerce problem domain is one that is highly user-centric, characterized by a multitude, enormity and wide mix of simultaneous user transactions (browing,searching and ordering). These operations place different sets of demands on the system used to provide the solution. For example, industrial catalogs may contain tens of thousands of products, each with several interrelated attributes. Performing high-speed searches on this amount of interrelated data might be an absolute requirement. Certain businesses require complicated calculations during order processing, such as computation of taxes and alteration of inventory, which requires high-speed access to time-series data and real-time computations on this data. The magnitude of customer and business' data is typically enormous.Secure transactions and order processing place stringent demands on data consistency and visibility.

ObjectStore and Javlin's unique Cache-Forward(TM) architecture for distributing data and processing provides one way to implement a distributed solution. Object relationships from the application space being stored as is in the database allows high-speed searches on interrelated data. ObjectStore's memory-mapped architecture allows in-memory speed computations on persistent data. Javlin provides distribution of cached persistent data that is required for scalability. With an already rapidly growing ObjectStore/Javlin customer base, the benchmark was implemented to characterize high speed and scalable architectures. The results are used to identify product deficiencies and performance bottlenecks that are likely to arise in typical development scenarios. Through this benchmark we can analyze the relative strengths of various configurations (for example, using different web servers or application servers) and make customer recommendations accordingly.

3 System Requirements

The workload specified in the TPC-W specification attempts to test the e-commerce environment for simultaneous on-line sessions via browsers, dynamic web page generation, consistency of web objects, simultaneous execution of a variety of types of online transactions, transaction ACID properties, data read and update contention and shopping cart persistence through sessions. The application is a retail store on the internet with customer browse, search and order scenarios. Starting at the home page of the store, the user (a customer or an administrator) searches and selects items, retrieves detailed information on them, puts them in a shopping cart, registers personal information, provides secure payment information, verifies a purchase and updates the item catalog. The TPC-W precisely specifies the pre- and post-conditions for all use-cases (or "web interactions"), primary database entity relationships, field definitions, database population requirements and data transparency rules. There also exist implementation restrictions and lower bounds on performance metrics.

4 Design Overview

We needed a flexible and extensible system that can be used to test various configurations. Given the requirements of an HTML front-end and HTTP for communication between the system and user, we decided to use a Java-centric architecture using JavaServer pages for presentation. To meet the scalability requirements we use the Javlin middle tier library for data distribution. Other than these fixed parameters, it was deemed desirable to have a system whose architecture can be changed with minimal modifications: for example, introducing another tier for scalability, or using a different web server or even a different database at the back end.

4.1 System Components

Figure 1 depicts the principal components of a general system we wish to develop with the above goals in mind. These include an HTTP server to handle requests for static pages and to dispatch requests for dynamic pages to the respective engine; a JSP Engine(s) to handle the generation of dynamic pages; an application server containing beans that handle business logic associated with processing of a particular incoming request; Javlin for distribution of data caches and scheduling of database transactions; and a database management system such as ObjectStore.

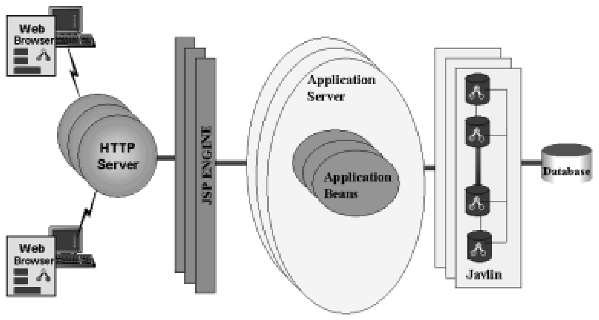


Figure 1: Figure 1: System Components

System scalability can be achieved by various strategies:
1. *HTTP Request Dispatching.* A network dispatcher route requests to multiple HTTP servers. Depending on the vendor, each web server can further route requests for dynamic content to multiple JSP Engines. Architectures that use sockets (eg. Apache/Jserv) or ISAPI/NSAPI (eg. Netscape IPlanet Server) for communication between the web server and the JSP Engine lend themselves toward this strategy while in others like WebLogic scaling would achieved by routing HTTP requests amongst a cluster of servers.
2. *Distribution of Processing.* Most JSP engines compile the JSP into a servlets that retrieve information needed for the presentation by invoking methods (local or remote) on application beans. Using remote beans (such as Enterprise JavaBeans) allows distribution and balancing of request load from the presentation servlets.
3. *Distribution of Data and Data Caches.* Using ObjectStore allows data to be distributed across multiple databases and data servers. Further, the cached data accessed by the application beans itself can be distributed using the Javlin Database transactional requests are routed to different data caches based upon the access type (read vs. update), nature of the interaction (particular subsystem accessed) and the existing transactional load of caches.

4.2 Implementation

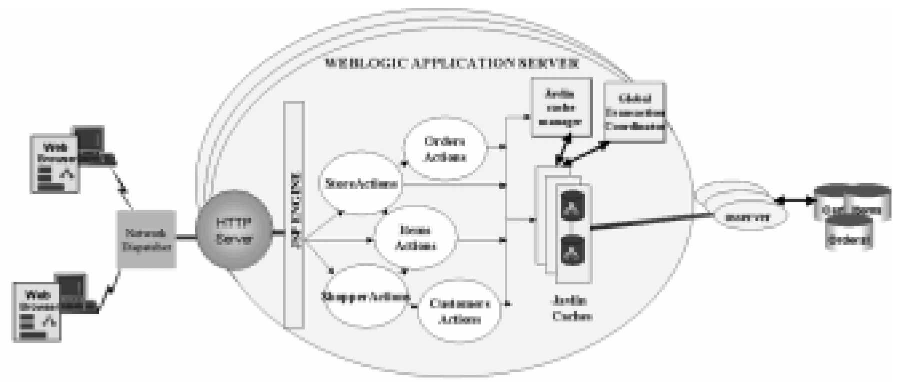


Figure 2: Components for particular implementation using WebLogic Server

The current implementation uses beans instantiated in the same Java Virtual Machine as the servlets. These along with the global transaction coordinator, JSP Engine and Javlin caches reside in the same VM as the WebLogic Server. System scalability is achieved by HTTP request dispatching, data and cache distribution.

Interactions fall in the following five categories. *Browse interactions* like displaying best sellers or latest products in a particular category, retrieving thumbnails for a random set of (promotional) items occur with very high frequency and require throughput. *Searching* items by criteria such as subject, author or title occur with low frequency, require moderate response time and have relaxed consistency. The *Shopping Cart interactions* involve both update operations on cart objects and read access to multiple types of data (customers and items) but requires very good response time. *Ordering interactions* create order objects with low frequency but require strict consistency and communication with an external payment gateway. *Administrative interactions* change information about items and users, happen very infrequently but are highly update and compute intensive. We list a few considerations in the database design to be able to achieve a reasonable mix of all the above interactions.
1.*Clustering.* Segregating objects in the database allows simultaneous access of unrelated data. For example, keeping shopping carts in different database clusters avoids contention during simultaneous updates to different carts.
2.*Using non-blocking reads.* We use the non-blocking read feature of ObjectStore (which guarantees consistent but not necessarily up-to-date data) whenever consistency is relaxed or can be achieved by application logic. For example, all search and browse interactions access items information with somewhat relaxed consistency. Creation of orders, on the other hand, needs to read latest product and customer information. However, since this data is rarely modified, we can attain a high read throughput by sendingnotifications to the reader cache on the occurence of an update. Creation of shopping carts also requires reading consistent information about items. However, an explicit check for consistency at various stages of the business transaction is possible since this needs involve only the determination whether the item is valid or not
3.*High concurrency collection representations.* Multi-tier collections for Order objects and Item objects (the latter are categorized by Subject, since there is a fixed set of subjects) prevents the container holding these objects to be a point of contention during creations and updates. This also allows distribution of data.
4. *Indexing.* All browse and search interactions make use of indexes on the desired criterion. While this allows faster retrieval, maintenance of these can be non-trivial. Some of these structures such as the index of related items and best selling items require being update on each order placement.
5.*Lazy Update via Inter-cache Messaging.* Defering of database updates (such as the index structures mentioned above) by sending messages across caches improves real-time response requirements. For example while placing orders, the information about the newly created order is notified to a cache that updates the index structures based on the latest orders.
6.*Data Replication.* Shopping carts replicate data about items and customers so that updates and reads of the cart (which happen more frequently than creation) does not require access to these objects. Since checks for consistency are made only during order creation (once per lifetime of a cart), synchronization of replicated data does not become an issue.
7.*Improving Transaction Throughput.* Batching of database read and update transactions (handled by Javlin) significantly improves transaction throughput unless atomicity requirements disallow so (such as the order creation). In the latter case we map business transactions to database transactions but make use of an application level locking to keep these transactions short.

5 Testing Environment

Our system is still under development though some preliminary tests have been conducted. To drive the sytem for high throughput and performance, a Remote Browser Emulator component (RBE) was implemented to simulate a stateful user browser. The results presented in the next section were obtained with the system under test running on an HP-NetServer machine (Four Pentium Zeon 500MHz CPUs, 1 Gb RAM) running Windows NT Server 4.0, WebLogic Server 4.51, Javlin 1.0 and ObjectStore 6.0. The RBEs ran on various NT and Solaris machines. The database was prepopulatd with 10K items and customers each, 2K orders and 500 shopping carts. Each measurement interval lasted for about two hours.

6 Current Results

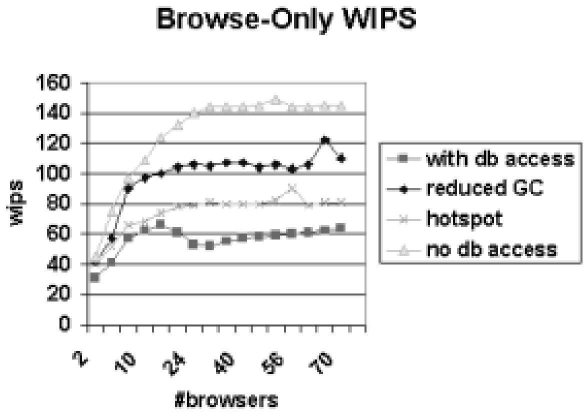


Figure 3: WIPS behavior, browse-only scenario

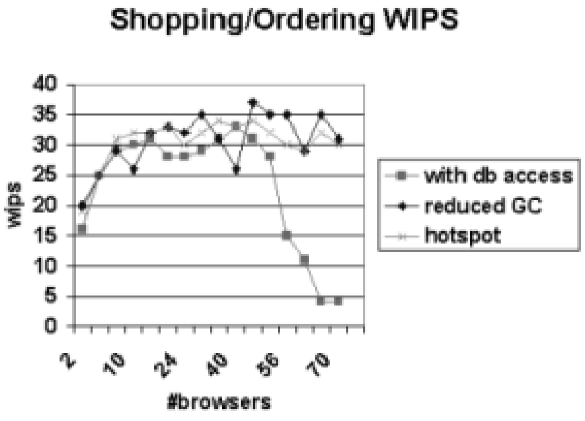


Figure 4: WIPS behavior, shopping/ordering

6.1 Performance Metrics

Response time for each interaction and throughput are sampled during each measurement interval every 30 seconds. Each test is performed for various simulated load profiles from the RBE. A load profile comprises of a particular browse/shopping/ordering mix and a number of browsers. We also recorded statistics for a system running with with no database access to provides us with a comparative basis for measuring ObjectStore and Javlin performance without webserver and JSP overhead.

Two metrics of interest are the response time (RT) for a particular interaction and the number of Web Interactions Per Second (WIPS) as a function of the number of browsers. These two metrics together measure the system under test for cpu utilization and concurrency behavior. While RT is used to characterize cpu utilization, the behavior in the saturation of the WIPS level when sufficiently large number of browsers hit a constant cpu-powered system determines system concurrency.

6.2 Preliminary Results

Figures 3 and 4 show the WIPS behavior for browsing and ordering profiles respectively. Profiling indicated that a lot of processing time was spent in garbage collection. Tests with using the HotSpot VM (which evens out the garbage collection process) and by tuning the system to drastically reduce the string objects indicate great improvement in overall throughput. The fact that the WIPS actually drops rather than saturates when there is no optimization for GC implies that excessive GC can severely hamper scalability.

From the response time results for single browsers (Figure 5) we see that, in fact, reducing GC drastically improves response times browse-only interactions in particular. In fact for some of these interactions there is hardly any difference in response time when there is no database access. However the amplified difference in the saturation level WIPS implies concurrency bottlenecks in the system.

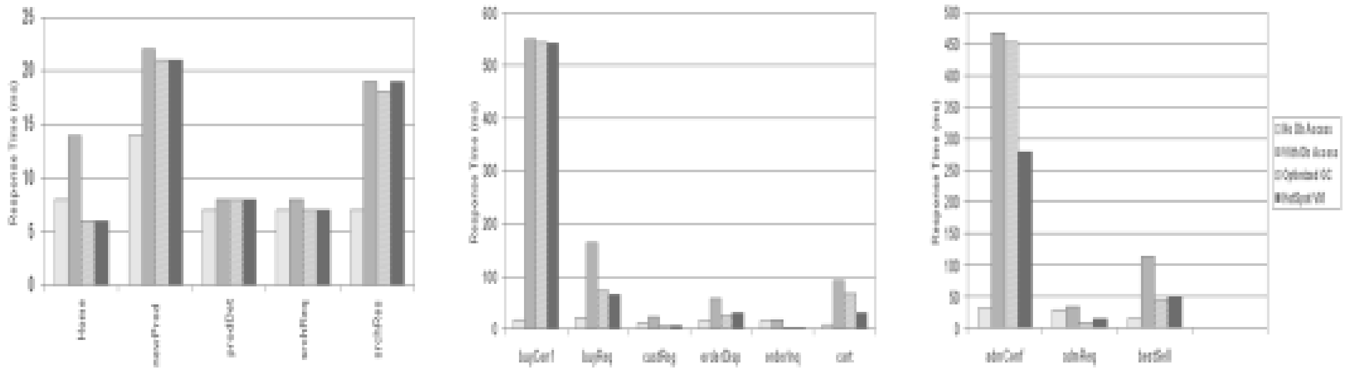


Figure 5: Response times (with single browser) for various interactions.

7 Summary and Conclusions

The TPC-W lays out a set of specifications that are typical in a real time e-Commerce application. We have built a system that satisfies these requirements using ObjectStore and Javlin. This paper outlines some of the design considerations while developing this system. In this process, we have illustrated how to develop a high performance and scalable system using various configurations. While results are still preliminary, their inclusion here illustrates how to use identify system bottlenecks for speed and concurrency. This helps in tuning performance of not only the implementation but also the product lines (ObjectStore and Javlin) themselves. Our goals from this implementation have been to direct the development of these products for being able to deliver high performance applications and to provide customer recommendations for specific architectures.

References

[1] Zona Research, *Second generation of electronic commerce* August 1999. $http://www-4.ibm.com/software/webservers/commerce/servers/secondwave.pdf$

[2] Transaction Processing Council, *TPC Benchmark W Draft Specification*, Revision D-5.1, August 1999

An Overview of the Agent-Based Electronic Commerce System (ABECOS) Project

Ee-Peng Lim, Wee-Keong Ng
Center for Advanced Information Systems
School of Applied Science
Nanyang Technological University
Nanyang Avenue, Singapore 639798, SINGAPORE

Abstract

ABECOS (Agent-Based Electronic COmmerce System) is an ongoing project that investigates the problems involved in building a large-scale business-to-consumer e-commerce system on the web using agent technologies. In this paper, we describe the ABECOS's system architecture and some research problems studied by the ABECOS research group. Our approaches to apply database techniques to address some of these E-Commerce problems will also be presented.

1 Introduction

In the 1998 NSF Workshop on Research Priorities in Electronic Commerce[4], e-commerce has been defined as "sharing business information, maintaining business relationships, and conducting business transactions by means of telecommunication network." Under this broad definition, e-commerce has further been classified into *business-to-business* and *business-to-consumer*. As pointed out by Ozsu in [10], business-to-business e-commerce activities started as early as in the 1960's in the form of Electronic Data Interchange (EDI) although the telecommunication network adopted by the EDI systems were not Internet-based. In the case of Internet-based EDI [3], the main research problems include the interoperability between different EDI systems and the efficient and secure transmission of EDI messages on the Internet.

The business-to-consumer e-commerce, in contrast, has only started to pick up recently. Nevertheless, within a short span of time, it has created many new ways of online retailing on the Internet and has contributed to the successes of many new ".com" businesses. For example, Amazon.com has now become one of the world's largest booksellers, boasted with a customer base of 12 millions[12]. Yahoo.com, besides providing a very successful web search engine, has been hosting 6000 over online stores selling more than 3.75 million different products. Despite the huge successes of these ".com" businesses, many issues in the business-to-consumer e-commerce remain unresolved and require the attention of database researchers.

In the ongoing ABECOS (Agent Based Electronic COmmerce System) project[1] at the Center for Advanced Information Systems (CAIS), we are focusing on research issues related to building a large-scale e-commerce system that facilitates Internet shopping using different kinds of agents[13]. ABECOS aims to support a distributed

[1] See *http://www.cais.ntu.edu.sg:8000/∼abecos/*

marketplace where large number of different products offered by different online stores can be purchased by web users.

The ABECOS system architecture essentially consists of three types of software agents, i.e. **seller agents**, **buyer agents** and **directory agents**. Each buyer agent represents a web user and performs the appropriate tasks to purchase some product(s) specified by the user. In its operations, the buyer agent will emulate some intelligent behaviors of a typical human shopper in its attempt to find the “best” deal for the user. Here, the measure of a deal could be determined by the price of the product(s) or the some other criteria specified by the user. To buy a product from an online store, a buyer agent has to communicate with the seller agent representing the store. The seller agent has direct access to the product database of its online store and is capable of committing transactions on behalf of the store. The buyer agent may negotiate with the seller agents of multiple stores before a final purchase decision is reached. To allow buyer agents to locate seller agents of the appropriate online stores, the ABECOS system architecture has incorporated directory agents that maintain catalogs of product information in a summarized form and are capable of recommending the seller agents of appropriate online stores to the buyer agents based on product specifications provided by the users[9].

In the remaining sections of this paper, we will highlight some database research issues in the context of ABECOS project.

2 XML Search Capabilities

XML enables online stores to provide their product information in a self-describing and platform-independent manner to the web users. By marking web documents with well-formed user-defined tags, online stores can easily structure the document content so that remote e-commerce applications can extract the embedded data for their use[8]. When XML documents are given to a web browser together with their XSL style sheets(XML Style Sheet Language), their XML content will actually be presented in HTML format within the browser. Hence, XML is able to support both the browsing and interoperability needs of e-commerce applications.

In ABECOS, we adopt an XML approach to store both product catalog and online store catalog information at the directory agent’s web site. Products are first classified into product categories and sub-categories forming one or more product hierarchies (also known as the *isa* hierarchies). The leaf nodes of the product hierarchies are XML pages of individual products while the internal nodes are XML pages containing directories of products or product categories. These XML pages are connected by XML links. The product hierarchies greatly facilitate web users in navigating the product catalog XML pages and formulating search queries on only products under some specific category. Beside the *isa* relationship between product categories and products, there also exists some other mundane or peculiar inter-relationships between products or product categories. For example, a personal computer consists of CPU, disk drives, and others as parts. A special mouse device can only be used with a special keyboard. All product information, product category information and the different types of relationships between them can be readily represented in XML. Similarly, we can maintain online store catalog information using XML. In the following, we describe our research efforts in developing a search engine for storing and indexing these XML information.

At present, our research in XML search capabilities focuses on providing **structured queries** on text data embedded in XML pages[5]. Here, structured queries are ones that involve **structure patterns** within the XML pages or across them. For example, a web user may want to look for an IBM personal computer consisting of an energy saving monitor and an ergonomic keyboard. Such requirement can be translated into a structured search on the “brand” element within “personal computer” pages that are linked to “monitor” pages containing the terms “energy” and “saving” in their “feature” elements, and “keyboard” pages containing the term “ergonomic” in their “feature” elements. This query can be represented as a query graph shown in Figure 1.

To efficiently evaluate the above type of queries on a collection of XML pages, we have developed a suite of indexes consisting of a **tag inverted index** and a **term inverted index**. The former allows us to locate the

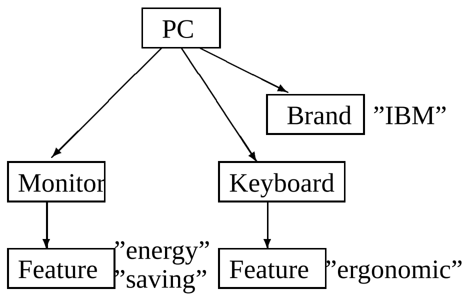


Figure 1: A Structured Query Example

start and end positions of tags within XML pages. The latter identifies the positions of terms within XML pages. Nevertheless, the indexes alone do not directly lead to efficient query evaluation if the structure patterns used in queries are not properly considered. Hence, coupled with these indexes, we have also developed an **incremental greedy** algorithm to decide the order of evaluating structure patterns in the queries. The algorithm consists of multiple iterations and in each iteration the query node expected to return the least result is evaluated. The algorithm terminates when it has evaluated all query nodes in the query graph. A prototype implementation of our proposed XML search engine has been made available on the web at *http://www.cais.ntu.edu.sg:8000/∼wires*.

3 Product Schema Integration

In the ABECOS e-commerce system, buyer agents and seller agents interact and transact on behalf of sellers and buyers while directory agents locate seller agents for buyer agents. Since seller agents distinguish themselves by the products they sell, directory agents use the buyer's specification of the desired product as a query to find relevant seller agents. In this case, directory agents no longer use a simple keyword search strategy adopted by traditional search engines; they use user-supplied product specifications as the search criteria. They compare the specifications with descriptions provided by different seller agents and return the locations of relevant seller agents. Once buyer agents obtain these addresses, they communicate directly with each relevant seller agent. They then query for the details of the products, negotiate prices and complete the deal.

As different sellers use different local product schemas, there is a need to resolve the semantic differences among local schema attributes. This is an ontology heterogeneity problem. Data integration is a solution to this problem. Product schema integration is a sub-field of data integration. In the Asilomar report on database research[1], in J. Gray's brief essay on database research [2] and in another paper on database research in the 21st century by A. Silberschatz *et al.*[11], heterogeneous data integration has been identified as one of the major research directions in the near future. It has also been pointed out that in e-commerce systems, heterogeneous information sources must be integrated and the integration is an extremely difficult problem. Although significant progress has been made in data integration, this problem is by no means solved and *name matching* across distributed sources is still one of the major problems remaining. Therefore, we need schema management facilities such as directory agents in ABECOS which can adapt to the dynamic nature of heterogeneous data. We may re-state the problem as a set of basic questions:

- What is the information needed for identifying the correspondence among attributes from heterogeneous local product schemas? In what form does such information exist?
- What is the algorithm to integrate product schemas? Is it capable of automated integration?
- How can the global schema be maintained?

We refer to this problem as the *Product Schema Integration* problem (PSI)[13, 14].

4 Electronic Brokering

In [7], *product brokering* and *merchant brokering* are two types of **electronic brokering (e-brokering)** to be carried out as part of the buying process. Product brokering refers to helping a web user to determine what product to buy. Merchant brokering involves helping the user to determine who to buy from. At this point, it is not always clear if product brokering always comes before merchant brokering. It is however quite certain that users will always have to start with some selection criteria on some product attributes, e.g. product type, brand, etc..

In our e-brokering research, we model the brokering of products and merchants as a *database selection problem*. In this database selection problem, we are given a set of user-specified selection criteria on some product attribute and we have to select from a large number of product databases from different online stores the ones that are likely to provide the products that satisfy the selection criteria. Once some online stores have been shortlisted by the user, the selection criteria can be forwarded to the shortlisted stores and be actually evaluated against the product databases of the stores. The products satisfying the selection criteria from the selected stores are finally combined and presented to the user for his final buying decision. Based on the above problem formulation, we seek to answer the following research questions in our e-brokering research:

- What should be the summarized knowledge one should acquire from the product databases such as database selection can be accurately performed? In this respect, the database selection technique must consider multiple attributes of different data types in the product databases.

- How do we measure the performance of database selection? The performance measure may possibly be determined by the end users.

5 Web Site Refreshing

As corporations advertise themselves and sell products through their websites, it is necessary to place the product and price information on web pages for consumer to browse. These information generally come from a corporation's product databases. For a large corporation with multiple websites spread across countries and continents, coordinating the refresh of information to websites from product databases is a complex task. There are several complications:

- A website contains information that come from different portions of the product databases. For instance, a single web page alone may contain different types and amount of information from different parts of the databases.
- Websites have different refresh frequencies. Pages in the same host machine may also have different refresh frequencies. Even different parts of the same page have different refresh frequencies.

In order to keep websites up-to-date, an appropriate set of queries must be made to product databases so that query results can be distributed to refresh the correct websites. Queries must be made regularly so that fresh information arrive within the refresh frequencies of websites.

A straightforward approach to refresh websites is to gather all the queries required by each website and execute each of them. However, this method may not work because query results may not arrive within the refresh frequencies of websites due to the lack of resources (processors). To alleviate this problem, we may use more computers and resources. However, this option is viable only if financial resources are available.

We observe a characteristic of information hosted in websites: As websites display information for consumers, care is taken not to overwhelm consumers with large amounts of complex information. Generally, information is distributed across multiple web pages and aesthetically presented. This has two implications:

- Fresh information from several web pages may be obtained by performing a single query to a corporation's product database.
- Likewise, a single query's result may be splitted up and distributed across multiple web pages, thereby satisfying their information requirements.

Given the information refresh requirements of a set of websites, we must find a suitable set of queries to pose to product databases so that *timely* refreshes are made to websites. Clearly, the query set should be minimal. We may re-state this problem as a set of basic questions: Given the information refresh requirements of a set of websites,

- How to determine the feasibility of scheduling a set of queries to satisfy them (i.e., query results arrive within refresh frequencies)?
- If the query set is non-unique, what is the optimal query set? What optimization techniques are there?

We refer to this kind of problem as the Websites Refresh Problem (WRP)[6]. This problem needs not necessarily occur for a large corporation with multiple geographically dispersed websites. It may also happen within a single, complex website containing multiple web pages requiring bits and pieces of information to be refreshed at different intervals from different portions of the corporation's product databases.

6 Concluding Remarks

In this paper, we gave an overall description of the ABECOS project and the database research problems involved. The demand for large-scale business-to-consumer e-commerce system has certainly brought about many new problems that require one or more solution techniques. While solving these research problems, we have started to develop a ABECOS system prototype to demonstrate the capabilities of buyer agents, seller agents and directory agents. To allow agent technology to be widely and quickly deployed by the online stores and web buyers, we are now looking into software toolkits that allow us to create and configure the different types of e-commerce agents.

References

[1] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The Asilomar Report on Database Research. *ACM SIGMOD Record*, 27(3), December 1998.

[2] J. Gray. Database Systems: A Textbook Case of Research Paying Off. URL=*http://www.cs.washington.edu/homes/lazowska/cra/database.html*.

[3] T. Harding, R. Drummond, and C. Shih. *Internet Draft: Requirements for Inter-operable Internet EDI*. EDI-INT Working Group, URL=*http://www.ietf.org/internet-drafts/draft-ietf-ediint-req-08.txt*, September 1999.

[4] IC^2 Institute. NSF Workshop: Research Priorities in Electronic Commerce. Technical report, The University of Texas at Austin, September 1998.

[5] E-.P. Lim, C-.H. Tan, B-.W. Lim, and W-.K. Ng. Querying Structured Web Resources. In *ACM International Conference on Digital Libraries*, pages 297–298, Pittsburgh, June 1998. ACM Press.

[6] H.-F. Liu, W.-K. Ng, and E.-P. Lim. Some Preliminary Results on the Feasibility of Website Refresh Queries. In *10th International Conference on Database and Expert System Applications (DEXA'99)*, Florence, August 1999.

[7] P. Maes, R.H. Guttman, and A.G. Moukas. Agents That Buy and Sell. *Communications of the ACM*, 42(3):81–91, March 1999.

[8] B. Meltzer and R. Glushko. XML and Electronic Commerce. *SIGMOD Record*, 27(4):21–24, December 1998.

[9] K.-L. Ong, H.-H. Chew, W.-K. Ng, and E.-P. Lim. Calling All Agents, Where Are You? In *International Workshop on Information Technologies for Electronic Commerce (ITeC'99)*, Florence, August 1999.

[10] M.T. Ozsu. Data Management Issues in Electronic Commerce. In *ACM SIGMOD International Conference on Management of Data*, page 505, Philadelphia, June 1999.

[11] A. Silberschatz, M. Stonebraker, and J.D. Ullman. Database Research: Achievements and Opportunities Into the 21st Century. *ACM SIGMOD Record*, 25(1):52–63, 1996.

[12] T. Wolverton. CNET News.Com: Amazon tries on hosting hat, September 29 1999. URL=*http://news.cnet.com/news/0-1007-200-266350.html*.

[13] G.-H. Yan, W.-K. Ng, and E-.P. Lim. Toolkits for a Distributed, Agent-Based Web Commerce System. In *International IFIP Working Conference on Trends in Distributed Systems for Electronic Commerce (TrEC'98)*, Hamburg, Germany, June 1998. Also appeared at the *Web Development Journal* at *http://WebDevelopersJournal.com/articles/ecommerce/yan01.html*.

[14] G.-H. Yan, W.-K. Ng, and E.-P. Lim. Incremental Maintenance of Product Schema in Electronic Commerce—A Synonym-Based Approach. In *IEEE 2nd International Conference on Information, Communications and Signal Processing (ICICS'99)*, Singapore, December 1999.

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398