IEICE TRANS. INF. & SYST., VOL.E84–D, NO.10 OCTOBER 2001

PAPER *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems and Systems' Assurance*

A Decentralized XML Database Approach to Electronic Commerce

Hiroshi ISHIKAWA[†] ***and*** **Manabu OHTA**[†]**,** ***Regular Members***

SUMMARY Decentralized XML databases are often used in Electronic Commerce (EC) business models such as e-brokers on the Web. To flexibly model such applications, we need a modeling language for EC business processes. To this end, we have adopted a query language approach and have designed a query language, called XBML, for decentralized XML databases used in EC businesses. In this paper, we explain and validate the functionality of XBML by specifying e-broker business models and describe the implementation of the XBML server, focusing on the distributed query processing.

key words: *decentralized database, query language, query processing, electronic commerce, XML*

1. Introduction

XML data are used in Web information systems and Electronic Commerce (EC) applications [15]. In particular, e-broker [11] business models on the Internet like Amazon.com, use a large number of XML data such as product, customer, and order data. In order both to flexibly model and agilely realize such applications, we need a modeling language for EC businesses, in particular, business processes. First, as business process specification, we must allow users to retrieve only necessary portions of XML data in EC businesses. Second, we must allow users to combine XML data from different sources in the Web to produce new XML data for EC services. To this end, we will adopt a query language approach to modeling EC businesses and will provide a query language for XML data centric in business models, tentatively called *XBML* (Xml-based Business Modeling Language) [8]. As a query language approach, we need to design a modeling language continuous with nonprocedural standards such as SQL.

We rationalize the necessity of a modeling language for EC businesses. First, the modeling language must be able to integrate the components by reducing their complexity and to make the integrated system understandable enough to claim the novelty. Second, the modeling language must be able to do more than model EC businesses. Thus, it must be able to bridge the gap between the constructed model and the realized system to survive agile competition in EC businesses.

Manuscript received March 23, 2001.
Manuscript revised May 23, 2001.

[†]The authors are with the Department of Electronics and Information Engineering, Tokyo Metropolitan University, Hachioji-shi, 192–0397 Japan.

Indeed, use of XML as interfaces of each component makes integration easy. However, this approach just models only the static aspects of the components. Instead, we must be able to model the dynamic aspects of business models, that is, business processes. For example, the author [3] takes an HTML/JavaScript approach to modeling Web-based applications in the context of extending UML. But this procedural approach would increase the complexity of modeling and the overhead of the client-server interaction on the contrary. Instead, we need a nonprocedural language approach to modeling the bushiness processes. However, just applying SQL to XML is inadequate because RDB and XML data have different data models. So we take a nonprocedural query language approach to modeling XML-based businesses. Further, we make an XML query language efficiently executable on the server-side to agilely implement the business models. In a word, our query language can model and realize XML-based EC businesses.

We will describe the functionality of XBML and validate the usability of XBML by specifying the example business model with XBML in Sect. 2. Next we will describe the implementation of an XBML server in Sect. 3. Finally, we will compare XBML with other works.

2. Approach

2.1 Database Schema and Business Model

We use the following database schemas or DTD fragments for illustrating the functionality of XBML:

```
<!ELEMENT bib (book*, article*)>
<!ELEMENT book (author+, title, publisher, price, keyword*)>
<!ATTLIST book year CDATA>
<!ELEMENT article (author+, title, publisher, keyword*)>
<!ATTLIST article year CDATA>
<!ELEMENT publisher (name, address)>
<!ELEMENT author (firstname?, lastname, office+)>
<!ELEMENT office (#PCDATA — (building, room))>
<!ELEMENT registration (register*)>
<!ELEMENT register (customer)>
<!ELEMENT customer (id, lastname, keyword*, purchased*)>
<!ELEMENT ordering (order*)>
<!ELEMENT order (id, item)>
<!ELEMENT shipping (ship*)>
```

```
<!ELEMENT ship (id, status)>
```

The following XML data conform to the above DTD:

```
<bib>
 <book year="1993">
    <author>
       <firstname>Hiroshi</firstname>
       <lastname>Ishikawa </lastname>
       <office>
          <building> L2 </building>
          <room> S210 </room>
       </office>
    </author>
    <title>Object-Oriented DatabaseSystem
     </title>
    <publisher> Springer Verlag </publisher>
    <price> 69.00</price>
 </book>
</bib>
```

We take an ordered directed graph as a logical model for an XML query language XBML as a modeling language of EC businesses. That is, the data model of the XML query language can be represented as data structures consisting of nodes (i.e., elements) and directed edges (i.e., contain, or parent-child relationships), which are ordered.

We also use e-broker business models based on XML data for describing the XBML functionality. Here we will provide the working definition to EC business models in general. The EC business models consist of business processes and revenue sources based on IT such as Web and XML. We assume that explanatory e-broker business models on behalf of customers consist of at least the following business processes (see Fig. 1):

(1) The customer searches products by issuing various (i.e., either precisely- or approximately-conditioned) queries against one or more suppliers and /or navigating through the related links.
(2) The customer takes recommendations from suppliers into account if any.
(3) The customer compares and selects products and puts them into the shopping cart.
(4) The customer checks out by placing a purchase order with registration.
(5) The customer tracks the order to check the status for shipping.

E-broker models can be modified to make other models such as auctions, which is not discussed here due to space limit. This means that the following discussion will be somehow valid to e-business models in general. The revenue source in e-broker models is sales. In general, the revenue sources itself requires further processes for valuation. However, we treat them as atomic processes and focus on the business process part of the model in the context of modeling languages.

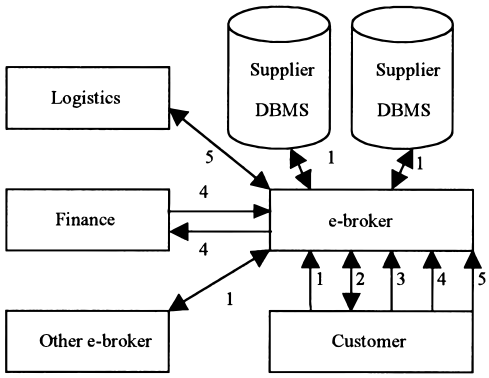

Fig. 1 E-broker model.

2.2 Business Model Specification

The authors [10] discuss that seemingly diverse business models can be represented by combining or modifying a limited number of commonly used business processes. The five processes described in the previous subsection correspond to such common processes. Further, we must take inter-enterprise (i.e., Business to Business) processes into consideration in describing business processes. In a word, the objective of XBML is to flexibly describe business processes involving autonomous distributed enterprise Web sites. Thus, we think that the validity of XBML can be shown if the previous processes are flexibly described by XBML. We describe the functionality of XBML and its usefulness by specifying the five processes of the e-broker models in succession. As business process specification requires a query facility for XML databases, we also discuss the validity of XBML from the viewpoint of XML query requirements proposed by W3C [18] when necessary.

2.2.1 Searching Products

The whole processes usually begin with a process for searching products. XBML provides the following functions to flexibly describe this process:

- XBML allows product search by *selecting* products based on their attributes, such as titles and authors, and *constructing* search results based on them.
- XBML allows ambiguous search by allowing *partially-specified strings* and *path expressions*.
- XBML supports *data join* used in related search to promote cross-sell and up-sell.
- XBML configures search results by *sorting* and *grouping* them based on product attributes.
- XBML supports "comparison model" of similar products by allowing search *multiply bound* across shopping sites.
- XBML provides localized views (e.g., prices) of global products by introducing *namespaces* (i.e., contexts).

All the functions but multiple binding conform to the core parts of W3C query requirements whose main purpose is XML data search. Thus, they are sufficient for describing search processes. Here multiple binding is prerequisite for business modeling although it is not included by the W3C query requirements. We will materialize the above in the rest of this subsection except sorting, grouping, and namespaces due space limit.

Data selection and construction

As the design of XBML has continuity of SQL to favor nonprocedural-ness, the syntax of XBML has the following basic structure although it is depicted in detail in Appendix:

select *{tag expression, ... , expression}*
from *d-variable* ***URI*** *uri ... uri, ...*
e-variable expression, ...
where *expression compare expression and*
expression compare expression or ...

Variables, called element variables, are declared in a from-clause. They are classified into variables directly bound to XML documents (i.e., d-variable) and variables bound to expressions (i.e., e-variable). Any order of variable declaration will do. This can provide XBML with nonprocedural-ness, which the W3C query requirements lack. An expression, called path expression, is denoted as a path consisting of one variable prefixed by "$" and followed by a series of tag names. In general, elements can contain multiple elements with the same tag name, which are basically ordered. The $(i+1)$-th element of *tag* is positioned by providing an index number i such as *tag[i]*. Attributes are referenced by prefixing "@" to them. The general syntax of path expressions has the following form:

$variable.tag... tag[i]... @attribute...

This can address arbitrary positions in hierarchical structures of XML data. Evaluation of an expression produces a set of values of the tail tag of the expression. In other words, a variable bound to an expression refers to such an element of a set at a time. In a where-clause, selection of XML data can be specified as logical combination of compared expressions. Compare operators include "=," "!=," "lt" for "<," "le" for "<=," "gt" for ">," and "ge" for ">=." The logical operators "not," "and," and "or" have precedence in decreasing order. New XML elements are constructed by combining expressions in a select-clause. The following query produces new elements consisting of titles and authors of books published by Prentice-Hall and firstly authored by Ullman:

(Query1)
select result{$book.title, $book.author}
from bib URI "www.a.b.c/bib.xml", book $bib.book
where $book.publisher.name = "Prentice-Hall"
and $book.author[0].lastname ="Ullman"
and $book.@year gt "1995"

In our current design, the resultant XML data have no DTD, that is, they are well-formed XML data automatically wrapped by a tag "XBML:result."

Partially-specified path expressions

First, we define semi-structured XML data as follows:

(1) Elements with the same tag are repeated at more than or equal to zero times, depending on parent elements, such as authors of books.
(2) Elements with the same tag have variant substructures, depending on parent elements, such as offices of authors.

These characteristics cannot be determined in advance. In XBML, a wildcard "%" can be included in expressions to specify such semi-structured XML data. In general, such an expression has the following structure:

$variable... tag_1.%.tag_2 ...

This can match with the following elements:

<tag_1>
<tag>
<tag_2>...</tag_2>
</tag>
</tag_1>

Through this functionality, XBML gains a merit to flexibly query semi-structured XML data whose structure is unknown in advance. The following query retrieves authors of any material such as book and article whose name is Ishikawa.

(Query2)
select result {$anyauthor}
from bib URI "www.a.b.c/bib.xml",
anyauthor $bib.%.author
where $anyauthor.lastname ="Ishikawa"

Data join

In XBML, the query for data join has the following syntax:

select *tag {expression$_1$, expression$_2$ }*
from *variable$_1$ ... , variable$_2$ ... ,...*
where *expression$_1$ ' = expression$_2$ '...*

Here expression$_i$ and expression$_i$' have either directly or indirectly common variable$_i$ declared in a from-clause. In a select-clause, new elements can be constructed by combining different elements, which are compared by the equality operator in a where-clause. Through this, XBML can provide a functionality corresponding to relational joins.

Partially-matched texts must be specifiable in comparison conditions to enable approximate search. In XBML, using a wildcard "%" in compared texts can specify the condition on expressions partially matching with "string" as follows:

(1) expression="%string%"
(2) expression="%string"
(3) expression="string%"
In particular, conditions (2) and (3) match with any text ending by "string" and with any text beginning with "string," respectively. The following query joins books and articles by authors as a join key within the same XML data:

```
(Query3)
select result {$article, $book}
from bib URI "www.a.b.c/bib.xml",
   article $bib.article, book $bib.book
where $book.author.firstname = $article.author.firstname
  and $book.author.lastname = $article.author.lastname
   and $book.title = "%Electronic Commerce%"
```

This facilitates related links on other articles written by authors of books about Electronic Commerce. Articles are promoted first-class products in the digital libraries. In e-broker business models, this helps increase cross-sell and up-sell. The query result has the following structure:

```
<XBML:result>
   <result>
      <article year="...">
          <author>...</author>
          <title>...</title>
          <publisher>...</publisher>
      </article>
        <book year="...">
           <author>...</author>
           <title>...</title>
           <publisher>...</publisher>
           <price>...</price>
       </book>
   </result>
   <result>...</result>
   ...
</XBML:result>
```

Multiple binding
In general, declaring a variable with multiple URIs in a from-clause specifies multiple binding as follows:

variable ***URI*** $uri_1, uri_2, \ldots, uri_n$

This binds the variable to multiple documents indicated by uri_i. Thus, the declared variable refers to elements of all the bound documents. As a result, XBML enables homogeneous retrieval of multiple data sources. This functionality is not included by the W3C query requirements, but is prerequisite for specifying inter-enterprise business processes. The following example retrieves books authored by the same author from two different online bookstores (bound to *bib*) by only a single query at the same time:

```
(Query4)
select result {$book.title, $book.author}
 from bib URI "www.a.b.c/bib.xml" "www.x.y.z/bib.xml",
    book $bib.book
where $book.author.lastname ="Ishikawa"
```

2.2.2 Recommendation

Recommendation processes compensate for searching processes in product selection. Related search is important in promoting cross-sell and up-sell, indeed. However, it is a technique *pulled* by the customer himself. As a more active technique *pushed* by the e-broker, recommendation is one of key solutions to increasing up-sell and cross-sell [14]. It is classified into three to the extent to which the customer is involved.

(1) *Non-personalized recommendation*
The customer is not involved. The e-broker recommends some products as general trends, independently of the customer. Or, the e-broker shows the customer products highly rated by the other customers.
(2) *Personalized recommendation*
The customer only is involved. The e-broker recommends some products based on the customer's psycho-graphic data, such as interests, or historical data, such as purchase records.
(3) *Collaboratively filtered recommendation* [14]
Both the customer and the others are involved. The e-broker recommends products purchased by those customers who purchased the products selected by the customer.

Function definition
We must specify all the three types of recommendation. We show that function definition and invocation of XBML can accomplish this. Here we focus on function definition and describe function invocation later. In general, the function definition has the following form:

functionname *'(' parameter-list ')'* ***as*** *'(' query ')'*

The body of functions is an XBML query itself. The result of function invocation is that of the specified query. The first type of recommendation can be facilitated by packaging a query as a function with no parameter, which corresponds to a relational view. The following function recommends recent EC books to the customers in e-broker business models as one of trends:

```
function Non-Personalized-Recommendation () as
(select result {$book.title, $book.price}
from bib URI "www.a.b.c/bib.xml", book $bib.book
where $book.keyword ="Electronic Commerce" and
   $book.@year gt "1998")
```

Using join of two XML data and a parameter in a function definition can facilitate the second type of recommendation. The following function recommends

products based on the keywords which the customer (specified by its identifier, *customerid*) have registered in advance as his psycho-graphic data:

```
function personalized-Recommendation (customerid) as
(select result {$book.title, $book.price}
from bib URI "www.a.b.c/bib.xml", book $bib.book,
   r URI "www.a.b.c/registration.xml",
     customer $r.register.customer
where $book.keyword = $customer.keyword and
   $customer.id = customerid)
```

Specifying join of two data, a parameter, and a condition on multiple-valued elements in a function definition can facilitate the third type of recommendation. The condition in this case is interpreted as true if at least one element satisfies the condition. The last example recommends products based on similarity that there are other customers who purchased the product selected by the customer (i.e., indicated by *selected*).

```
function collaboratively-filtered-Recommendation (selected)
as
   (select result {$book.title, $book.price}
   from bib URI "www.a.b.c/bib.xml", book $bib.book,
      r URI "www.a.b.c/registration.xml",
      customer $r.register.customer
   where $book = $customer.purchased and
      $customer.purchased = selected)
```

Function definition and invocation is recommended by the W3C query requirements [18].

Query transformation

Until now, we have treated recommendation and search as separate processes. However, searching can be enhanced to do the same effect as recommendation. Thus, when the customer specifies search keywords, the search result can be expanded to include recommended products by transforming the original search query. Query transformation for recommendation is classified into two rules as follows:

(1) *Keyword addition rule*
This rule has the general form:

keyword1 ==> keyword1 | keyword2

Transforming the rule into disjunctive conditions (i.e., "or") of an XBML query can accomplish this. For example, the originally specified keyword "Electronic Commerce" adds a new keyword "Internet Business" and the disjunctive condition (i.e., "or") is added to the end of the query as follows:

```
(Query5)
select result {$book}
from bib URI "www.a.b.c/bib.xml, book $bib.book
where $book.keyword = "Electronic Commerce" or
     $book.keyword = "Internet Business"
```

This technique is similar to query expansion [2] used in information retrieval. Note that this type of transformation keeps data sources unchanged.

(2) *Data source addition rule*
This rule uses set operations on queries to modify the original one. The rule has the following general form:

query1 ==> query1 set-operator query2

Transforming the rule into a set-operator "union" of XBML queries can facilitate this. For example, when the customer searches books on EC, he will automatically search articles on EC at the same time by modifying the original query with a disjunctive query to increase cross-sell in multiple product categories as follows:

```
(Query6)
select result {$book}
from bib URI "www.a.b.c/bib.xml, book $bib.book
where $book.keyword = "Electronic Commerce"
union
select result {$article}
from bib URI "www.a.b.c/bib.xml, article $bib.article
where $article.keyword = "Electronic Commerce"
```

Indeed, the basic functionalities such as disjunctive conditions and set-operators are recommended by the W3C query requirements, but transformation itself must be dynamically done.

Of course, the query transformation technique depends on creation of good rules. The manual rule creation is important in that it directly depends on the top-down marketing strategies themselves. Here, technically, we describe only the automatic rule creation based on Web mining techniques. In general, Web mining is categorized into Web content mining and Web usage mining [4]. Web content mining automatically structures information from Web contents. Web usage mining is based on user access data collected through the Web. So it is relevant to our context. Web usage mining is further categorized into Web-server-based one and application-based one. Most Web servers automatically generate and store access logs of pages while EC applications collect user access patterns such as a combination of keywords specified in a single query, a combination of product categories with keywords in a single Web session, and categories of products purchased in the past. So we analyze the application-based access patterns by using association rules and sequential patterns techniques [4] to create the transformation rules, not discussed in the paper due to space limitation.

2.2.3 Moving to Cart

As the third process, products selected as candidates for shopping are moved to shopping carts. To this end, temporary XML data must be accessed as well as permanent data. In XBML, this can be done by directly

referencing just retrieved data as follows:

$XBML:result.tag...

It is prerequisite for describing business models as combined processes, which is not included by the W3C query requirements. It enables sharing temporary data among successive processes. The following XBML query moves only the customer-checked items in the search result to the shopping cart:

(Query7)
select cart {item $result.book}
from XBML:result URI "www.a.b.c/XBML:result.xml,"
result $XBML:result.result
where $result.checked ="yes"

2.2.4 Placing Orders

This is the fourth process, which can be facilitated by assigning the result of a function invocation to an XML attribute and inserting it to permanent data. In XBML, attribute assignment with actual parameters given to a function can be specified as follows:

@attribute = function(parameter, ..., parameter)

Further, the following syntax is used for inserting values to existing data in XBML:

insertinto *target query*

This is not recommended by W3C query requirements, but is prerequisite for describing practical business processes. The following query places a purchase order in e-broker business models by consulting the current shopping cart and customer data and invoking a function:

(Query8)
insert into order
select order {@id = OrderID($customer.id, date()),
item $cart.item}
from r URI "www.a.b.c/registration.xml",
customer $r.register.customer,
XBML:result URI "www.a.b.c/XBML:result.xml",
cart $XBML:result.cart
where $customer.lastname = "Kanemasa"

Here, in a select-clause, function calls "OrderID ($customer.id, date())" generate unique order numbers. Ordering initiates internal processes, such as payment and shipment, hidden from the customers.

2.2.5 Tracking Orders

This completes business processes. It requires the facility for joining data distributed over multiple Web sites. XBML can accomplish this by the following general syntax:

select *tag {expression$_1$, expression$_2$}*
from *variable$_1$* ***URI*** *uri$_1$, variable$_2$* ***URI*** *uri$_2$,...*
where *expression$_1$' = expression$_2$'...*

Here expression$_i$ and expression$_i$' have either directly or indirectly common variable$_i$ bound to separate XML documents. Therefore, this is different from data join within the same document described previously. This is not recommended by the W3C query requirements, but is prerequisite for describing inter-enterprise business processes. The following query produces a set of ordered items and shipping status by joining order identifiers of order entry data and order shipping data at different sites indicated by separate variables such as *o* and *s*:

(Query9)
select result {$order.item, $ship.status}
from o URI "www.a.b.c/ordering.xml," order $o.order,
s URI "www.d.e.f/shipping.xml," ship $s.ship
where $order.id=$ship.id

3. Implementation

XBML is intended for use in not only modeling EC business models, but also realizing them agilely. Thus, XBML itself must be efficiently implemented and executed. We conclude this paper by describing the implementation and some feedback from the experiences.

XBML containing URIs intrinsically requires distributed query processing. So we construct the XBML server as follows:

(1) We construct local XBML servers as a basis.
(2) We construct global XBML servers by extending the local servers with server-side scripting techniques [17].

We describe the local and global servers in the following subsection in detail.

3.1 Local Server

We describe the basic architecture and implementation of a local XBML server. First, we describe storage schema for XML data. We have explored approaches to mapping DTD to databases (RDB, i.e., Oracle and ODB, i.e., Jasmine [7]) and to implement an XBML processing system [9]. If any DTD or schema information is available, we basically map elements to tables and tags to fields, respectively. We call this approach *DTD-dependent mapping*, where the user must specify mapping rules individually. Otherwise, we take a DTD-independent mapping or *universal mapping* approach (see Fig. 2), which divides XML data into nodes and edges of an ordered directed graph and stores them into separate tables for nodes and edges with neighboring data physically clustered. We provide separate tables for nonleaf and leaf nodes. The *order* fields of Leaf_Node and Edge tables are necessary for providing

Edge

Identifier	Order	Label_identifier	Child_Node_ identifier
Edge_Id_value ...	*Number* ...	*Label_Id_value* ...	*Node_Id_value* ...

Nonleaf_Node

Identifier	Path_identifier
Node_Id_value ...	*Path_Id_value* ...

Leaf_Node

Identifier	Order	Value
Node_Id_value ...	*Number* ...	*Value* ...

Attribute_Node

Identifier	Label_identifier	Value
Node_Id_value ...	*Label_Id_value* ...	*Value* ...

Fig. 2 DTD-independent mapping.

access to ordered elements by index numbers. Identifiers, such as ID and IDREF, realizing internal links between elements are declared as attributes and are stored as Value of the separate Attribute_Node table. So references are efficiently resolved by searching node identifiers in Attribute_Node.

We cluster data in node and edge tables on a breadth-first tree search basis to reduce I/O cost. Further, we have known from our preliminary experiments that the DTD-dependent mapping approach is mostly two times more efficient than the universal mapping. However, we prefer to the universal mapping for the following reasons:

(1) The approach can free the burden of defining idiosyncratic mappings from the users.
(2) The approach can store XML data whose DTD are unknown in advance.
(3) The approach can store heterogeneous XML data, i.e., semi-structured XML data in the same database.

Next, we describe the system architecture for a local XBML server or an XBML processing system (see Fig. 3). We make indices on tag values, element-subelement relationships, and tag paths in advance. We have found from our empirical study that the multi-key indices such as (identifier, path identifier) for nonleaf nodes are better than alternative single-key indices such as (identifier) in our current system because of higher selectivity.

We describe how the XBML processing system works. The XBML language processor parses an XBML query and the XBML query processor generates and optimizes a sequence of access methods for efficient execution. For example, query4 is parsed into a logical query plan represented as an ordered-graph, corresponding to the query semantics:

(Proj (Sel $book (Op_EQ $book.author.lastname "Ishikawa")) $book.title, $book.author)

Here, Sel, $Proj$, and $Join$ (not in the above example) denote selection, projection, and join of XML data, respectively. *Op_EQ* denotes "=." In our current implementation of the universal mapping, the primitive access methods are basic operations on node sets, executed by the XBML execution engine. We have known that both RDB and ODB are usable as the underlying database systems of the XBML execution engine with the upper layers unchanged only if the same set of the primitive access methods is dedicated to XBML execution engine by using the underlying database systems. We have implemented Oracle and Jasmine versions of XBML processing systems to confirm this fact.

3.2 Global Server

Now we construct the global XBML server by extending the above local XBML servers with server-side scripting

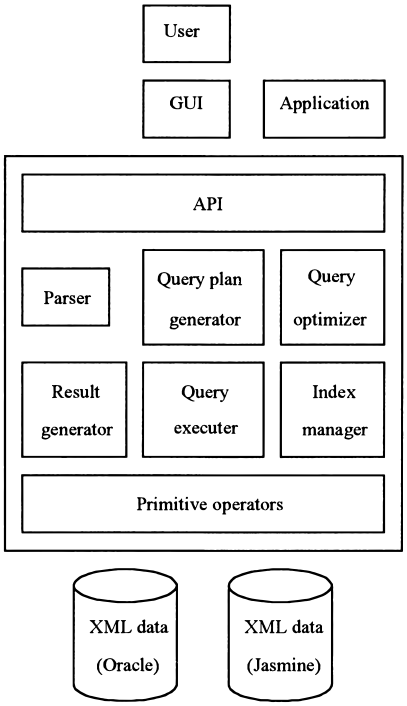

Fig. 3 XBML processing system.

techniques.

We provide preliminary definitions to queries. First, we categorize queries as follows:

(A) *Single-URI query*
This type of query contains only one XML data source specified by a single URL in the query, such as *Query1 (selection)* and *Query3 (join)*.

(B) *Multiple-URI query*
This type of query contains multiple XML data sources specified by multiple URIs in the query. This type is further categorized into two as follows:

(B1) *Decomposable query*
This type of query can be decomposed into a combination of single-URI queries with set operators, such as *Query4 (multiple binding)* and *Query6 (set operators)*.

(B2) *Non-decomposable query*
This type of query cannot be decomposed into a combination of single queries alone. This type of query contains join queries over multiple URIs, such as *Query9 (join of multiple data sources)*.

Second, we categorize queries in another way:

(a) *Local query*
XML data sources specified by URI are inside the relevant XBML server.

(b) *Global query*
XML data sources specified by URI are outside the relevant XBML server.

Now we show that non-decomposable (i.e., intrinsically global) query can be transformed into a series of single URL local or global queries and local queries (join). We assume that the original query contains n URIs. We translate a non-decomposable query by two steps:

(1) create a single-URI (local or global) query for each of n URIs with the insertion of the query result into the local server.
(2) create single-URI queries performing join of the results stored in the local server, which are local queries, by reducing all URIs to a single-URI.

Queries generated by the step (1) localize single-URI global queries. Of course, single-URI local queries remain local. We call them localized single-URI queries. After that, queries generated by the step (2) simulate join of multiple data sources by join of local data sources. We call them localized join queries.

Now we describe the global query processing, assuming that a query Q with a uri *URI* is specified as the input:

```
if Q is a single-URI query then
    process-or-dispatch (Q);
else {/*i.e., Q is a multiple-URI query; */
   if Q is a decomposable query then
      {for each sub-query Qsub in Q
          process-or-dispatch (Qsub);
       merge the result by the local server;}
      else {/*i.e., Q is not a decomposable query;*/
       decompose Q into localized single-URI queries
           Qloc-s and localized join queries Qloc-j;
        for each sub-query Qsub in Qloc-s
            process-or-dispatch (Qsub);
        process Qloc-j by the local server;}
      }
  process-or-dispatch (Q) /* for single-URI query*/
   {if URI is local to the server then
      process Q by the local server;
    else {/*i.e., Q is not local to the server; */
      dispatch Q to the relevant remote server;
      store the result into the local server;}
   }
```

The above query processing has some room for improvement in performance. Thus, if the non-decomposable query has no selection conditions, the whole remote data sources specified by the generated single-URI queries must be copied to the local server. So we refine the process-or-dispatch scheme to sort the result of the query and return the value range with respect to the join key (i.e., MIN and MAX values) by adding "order-by" to the query. The value ranges are kept as follows:

<result MIN=min-value MAX=max-value>...</result>

Then, the conditions "*join-key ge min-value and join-*

key le max-value" are dynamically added to the subsequent generated single-URI query. In turn, the query is evaluated to produce a new value range of the join-key (i.e., min-value' and max-value'). The following important characteristic holds between new and old value ranges:

min-value' > = min-value & max-value' < = max-value

From this, we can conclude that the expected selectivity is better than that of the original algorithm.

XBML works as server-side scripting with database access such as CFML [1], JSP [16], and provides universal access to distributed XML data. If XBML queries are embedded in XML-based scripts, the global XBML server can provide more direct and universal interfaces to representing and accessing distributed XML data than the other approaches. That is, XML pages containing the element <XBML> *XBML-query* </XBML> are interpreted as scripts. For example, an embedded query can be formulated to refresh up-to-date prices of products on demand access by joining product catalogues and pricing data distributed over the Internet.

4. Conclusion

We have analyzed the requirements for modeling EC businesses and have proposed and validated XBML as an XML query language approach to specifying EC business models. We conclude this paper by summarizing our contribution through comparison with other work.

There is no previous work on analyzing the requirements of modeling the business logic of E-businesses from the viewpoint of query languages to our knowledge. There are no high-level language approaches to modeling EC business processes. In particular, there is no other work on validating the modeling language by applying it to EC business models. An approach of extending UML [3] is incapable of describing the business logic non-procedurally. XBML can provide a more direct and universal tool for modeling distributed XML data-intensive EC applications than server-side scripting tools with SQL-based database access, such as CFML [1], and JSP [16]. Further, there is no other work on recommendation based on parameterized views (i.e., functions) and query transformation rules.

Now we will compare our XBML with other query language proposals although they are not principally intended for modeling Web-based EC businesses. Some features, such as multiple binding, function definition and use, set operators, and insertion, are crucially important in constructing EC business models. XML-QL [5] has functionality in common with our XBML. Unlike XBML, XML-QL lacks insertion and set operations. Condition specification with XML-QL is rather verbose. That would make query formation rather complex and decrease much efficiency in business process description. XQL[12] has common functionality with XBML. Unlike our XBML, however, XQL lacks construction, join, string regular expressions, orderby, groupby, multiple data source join, multiple binding, and insertion although it focuses more on filtering a single XML document by flexible pattern match conditions similar to XSL. XQL would require application logic in addition to query formation and decrease much efficiency in business logic description involving multiple sites. Lore [6] provides a powerful query language for retrieving and updating semi-structured data based on its specific data model OEM. However, it lacks functionality such as multiple binding unlike XBML. XQuery [19] is being designed as a standard query language for XML satisfying the W3C query requirements strictly. XQuery borrows features from its predecessors such as XQL, XML-QL, and Lore. XBML has an objective to flexibly specify business processes while XQuery has another objective to provide a facility for querying XML data. However, XBML includes a query facility as a part of specification functionality for business processes. Both of them have selection and construction, partially-specified path expressions, data join, and function definition and invocation in common. However, XQuery currently lacks temporary data access and permanent data insert as functions necessary for description of practical business processes. Further, XQuery lacks multiple binding and data join across multiple Web sites as functions necessary for specifying inter-enterprise processes. This difference is brought about the fact that XBML is mainly intended for specification of processes involving autonomous distributed data sources while XQuery is intended for "single-site" querying. Further, XBML can be used in the context of three-tier Web computing, which consists of Web clients, Web servers, and database servers. That is, an XBML query is used as a part of server-side scripting; The script embedding XBML queries requested by the client is evaluated to generate HTML/XML pages containing the query results at server-side and the generated pages are returned to the client. On the other hand, XQuery doesn't consider such usage currently.

Lastly, we describe our basic architecture about the security of XBML although it is not fully implemented. First, we utilize a public key infrastructure (e.g., Netscape's SSL [13]) for authentication of users and secured transmission of data (e.g., query and result) although the approach is not novel. More importantly, we must provide a scheme for authorization of access to data by users. Thus, we must provide individual users with different access rights for different parts of data. We provide *read* (select), *write* (insert), and *execute* (function invocation) as access rights. That is, we allow users to read/write data and to execute functions. Further, XML data being accessed have a hierarchical structure and accessed parts range from whole

documents to individual element values. We must take this into consideration when we design security management. So we propose a hierarchical access right control scheme as follows.

In general, access rights for elements with regard to users are specified as a UEA tuple:

{user element access-right}

This grants an access right for an element to a user. Any sub-element contained by the specified element is assigned the same access-right with respect to the specified user. This scheme is called access right inheritance. For example, if a whole document is specified as an element, any element within the document is assigned the same access right. If we intend to grant different access rights for different elements, we have only to specify different UEA tuples individually.

However, there are some cases where we want to provide some elements with an access right different from that provided to their super-elements. In such a case, we have only to add a new UEA tuple. This gives a new access right applicable only to a newly specified element with regard to the user. Of course, the other parts remain to have the old access rights with regard to the user. This partially modifies inheritance of access rights. For example, if *write* is granted for a particular element when *read* is already granted for its super-element, then this particular element is assigned *read* and *write* as a result. If we want to replace existing access rights by new ones, we have only to prefix access rights by "*" as follows:

*{user element *access-right}*

Further, in some cases, we must address parts of elements more flexibly. They include the following cases :

(1) Elements with the same tag are assigned separate access rights depending on values.
(2) Elements with the same tag are assigned separate access rights depending on positions in an ordered set.
(3) Combined elements are assigned the same access rights.

Using the function definition facility of XBML can flexibly do this. First, we define a function specifying accessed elements by a query. Then we provide UEA tuples with regard to the function as follows:

{user function execute}

Complete implementation of the security framework is our future work.

References

[1] Allaire Corporation, CFML, http://www.allaire.com/documents/cf4/CFML_Language_Reference/contents.htm, 2000.
[2] C.H. Chang and C.C. Hsu, "Enabling concept-based relevance feedback for information retrieval on the WWW," IEEE Trans. Knowledge & Data Eng., vol.11, no.4, pp.595–609, 1999.
[3] J. Conallen, "Modeling Web application architectures with UML," Commun. ACM, vol.42, no.10, pp.63–70, 1999.
[4] R. Cooley, B. Mobasher, and J. Srivasta, "Web mining: Information and pattern discovery on the World Wide Web," Proc. 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), 1997.
[5] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, XML-QL: A Query Language for XML, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, 1998.
[6] R. Goldman, J. McHugh, and J. Widom, "From semistructured data to XML: Migrating the lore data model and query language," Proc. 2nd Intl. Workshop on the Web and Databases (WebDB '99), 1999.
[7] H. Ishikawa, Y. Yamane, Y. Izumida, and N. Kawato, "An object-oriented database system jasmine: Implementation, application, and extension," IEEE Trans. Knowledge & Data Eng., vol.8, no.2, pp.285–304, 1996.
[8] H. Ishikawa, K. Kubota, and Y. Kanemasa, http://www.w3.org/TandS/QL/QL98/pp/flab.doc, 1998.
[9] H. Ishikawa, K. Kubota, Y. Noguchi, K. Kato, M. Ono, N. Yoshizawa, and Y. Kanemasa, "Document warehousing based on a multimedia database system," Proc. IEEE 15th Intl. Conference on Data Engineering, pp.168–173, 1999.
[10] S. Jones, M. Wilikens, P. Morris, and M. Masera, "Trust requirements in e-business," CACM, vol.43, no.12, pp.80–87, 2000.
[11] D. Jutla, P. Bodorik, C. Hajnal, and C. Davis, "Making business sense of electronic commerce," IEEE Comput., vol.32, no.3, pp.67–75, March 1999.
[12] J. Robie, J. Lapp, and D. Schach, XML Query Language (XQL), http://www.w3.org/TandS/QL/QL98/pp/xql.html, 1998.
[13] A. Rubin and D. Geer, Jr., "A survey of Web security," IEEE Comput., vol.31, no.9, pp.34–41, 1998.
[14] Special Section, Recommender Systems, Commun. ACM, vol.40, no.3, pp.56–89, 1997.
[15] Special Section, Web Information Systems, Commun. ACM, vol.41, no.7, pp.78–118, 1998.
[16] Sun Microsystems, JSP, http://java.sun.com/products/jsp/index.html, 2000.
[17] R. Vetter, "Web-based enterprise computing," IEEE Comput., vol.32, no.5, pp.112–116, 1999.
[18] W3C, XML Query Requirements, W3C Working Draft, 2000.
[19] W3C, XQuery: A Query Language for XML, W3C Working Draft, 2001.

Appendix: The Syntax of XBML

query = *select target* from *context-list* [*where-clause*] [*orderby-clause*] [*groupby-clause*]
target=expression | *tag* '{' *expression-list* '}'
expression-list = *expression* ',' *expression-list* | *expression*
expression = [*tag*] '$' *variable* | [*tag*] '$' *variable* '.' *path*
path = '%' | *tag* | '@' *attribute*| *path* '.' *path* | '(' *path* '|' *path* ')' | *text*
context-list = *context* ',' *context-list* | *context*
context = *variable* URI *uri-list* | *variable expression*
uri-list = *uri uri-list* | *uri*
where-clause = where *condition*

condition = *term* | *condition* or *term*
term = *factor* | *term* and *factor*
factor = *predicate* | not *predicate*
predicate = *expression compare expression*
orderby-clause = orderby *expression-list*
groupby-clause = groupby *expression-list*
insertion = insert into *target query*
function-definition = function *name* '(' *parameter-list* ')'
as '(' *query* ')'
parameter-list = *parameter* ',' *parameter-list* | *parameter*

Hiroshi Ishikawa received the B.S. and Dr.Sc. degrees in Computer Science from the University of Tokyo in 1979 and 1992, respectively. He worked for Fujitsu Laboratories Ltd., from 1979 to 2000. He is now a professor of the Department of Electronics and Information Engineering at Tokyo Metropolitan University. His research interests include databases systems, Web information systems, and e-commerce. He has published actively in international, refereed journals and conferences, such as ACM TODS, IEEE TKDE, VLDB, IEEE ICDE. He authored a book entitled *Object-Oriented Database System* (Springer-Verlag). He received the Sakai Memorial Distinguished Award from IPSJ in 1994 and the Director General Award from Science and Technology Agency in 1997. He is a member of IEEE, ACM, and IPSJ.

Manabu Ohta received the B.E., M.E. and Dr. Eng. In Electrical Engineering from the University of Tokyo in 1994, 1996 and 1999, respectively. He is a research associate of Tokyo Metropolitan University. His research interests are information retrieval and its Web application systems. He is a member of IPSJ.