A B2B Replication Service

A. Córdoba, J. Villadangos, J.J. Astrain, J.E. Armendáriz
Dpt. Matemática e Informática
Universidad Pública de Navarra
Campus de Arrosadía
31006 Pamplona (SPAIN)
alberto.cordoba@unavarra.es

Abstract

B2B transaction information can be stored as XML documents in RDBMS geographically distributed. In order to support fault-tolerance and persistent information storing, a deadlock free replication service is proposed in this paper. The service provides high available document retrieval against possible server failures, granting the access to the last version of the documents. It supports failures at the local manager level and at the repositories (RDBMS). The service also allows performing queries over the persistent repositories in order to recover previous commercial transaction information. Making use of this service, we propose a CORBA based centralized replication manager that ensures asynchronous passive document replication in terms of XML documents over a set of geographically distributed repositories.

Keywords: persistent data storing, replication service, XML, B2B, RDBMS, CORBA.

1. Introduction

B2B is one of the major concerns playing on the minds of corporate executives the world over. In interview after interview, senior managers say one of the biggest challenges facing their organizations is how effectively leverage the Internet, to give them a competitive advantage [24].

B2B applications extend market possibilities for companies offering such service. These applications consider a complex interaction between different technologies. Users perform commercial transactions with B2B applications using web interfaces. Interfaces interpret XML documents that contain the structure and data of commercial transactions. B2B applications track transactions and store persistently their data on databases. Moreover, B2B applications require a high degree of fault tolerance in order to maintain accessible all the information stored in the database.

In this paper, in order to consider the state of the art in fault tolerant B2B applications, we describe replication approaches applied for them, and we review the paper of the XML standard in such applications. The main contribution of the paper is to propose a replication service for replicated and geographically distributed B2B transactions.

There is no doubt that Internet offers nowadays more possibilities as ever. One of the problems that often appear is server failure and the chance the network brings, as a big resources repository, to recover the information that has been lost. There are works that are in charge of object replication in Content Distributed Networks (CDN), but they consider Internet nodes with infinite object storing capacity [16]. An object replication algorithm taking into account the variables that Internet has (Adaptive Data Replication - ADR) can be found in [23]. Another dynamic algorithm using a dissemination tree for replica establishment can be found in [4]. Other dynamic algorithms are in [29, 15]. Algorithms for replica management applying the Fault Tolerance Attribute Grammar (FTAG) are in [6, 5]. Other developed systems address issues related to object replication and fault tolerance in the context of CORBA. The EventService in CORBA does not offer acceptable results as a valuable object replication service [10]. As a more effective approach, the service is proposed as an interface to be used for replication. However, as stated in [14], in order to obtain more efficiency it is necessary to use node grouping mechanisms and the Object Group Service. In both cases, algorithms use multicast (push model) to obtain system information and/or suppose liable peer-to-peer connections. The Object Group Service (OGS) [11] provides replication for CORBA applications through a set of CORBA services. Replica consistency in ensured through group communication based on a consensus algorithm implemented through CORBA service objects. OGS provides interfaces for detecting the liveness of objects, and mechanisms for
duplicate detection and suppression, and for the transfer of application-level state. This approach has been followed by different works, which use the OGS or base their group communication service on it [28, 7, 19, 21, 18]. However, this implies that the Object Request Broker (ORB) should be modified. In [20], an alternative based on an interception mechanism is proposed. Its major advantage is that it does not require the ORB modification.

The extensible markup language (XML) has recently emerged as the new standard in the role of Internet information exchange [3]. The data of each commercial transaction of an B2B system has associated some information stored as XML documents, because this tag language is well suited to map it on Relational Database Management Systems (RDBMS) and to look for specific information in them. Many authors consider that XML schemas are persistently stored in Database Management Systems (DBMS). Many works can be found for RelationalDBMS [13, 25] (mapping XML structures into tuples) [9, 12], Object-RelationalDBMS [26, 17] (mapping XML structures into objects), applications in FederatedDBMS [8] and mixed solutions [27]. The XML Query working group has already released a query language XQuery, data model Xquery 1 and Xpath 2.0 Data Model and query algebra XQuery Formal Data Model [1]. Many and recently articles deal with XML queries in RDBMS [31, 22].

There are two approaches to designing database schemas for XML documents: Structure-mapping approach and Model-mapping approach. In the first one, database schemas represent the logical structure (or DTDs if they are available) of target XML documents. In a basic design method, a relation or class is created for each element type in the XML document. A more sophisticated mapping method has also been proposed, whereby database schemas are designed based on detailed analysis of DTDs [25]. In this approach a database schema is defined for each XML document structure or DTD. In the second one, database schemas represent an XML pattern. In this approach, a fixed database schema is used to store the structure of all XML documents.

An example includes the edge approach concept [13], in which edges in XML document trees are stored as relational tuples. Both database schema design approaches consider XML documents as a set of fragments composed of logical units. Obviously, these composition approaches have drawbacks –it takes time to restore the entire or large portion of the original XML documents, and processing certain text operations such as a proximity search beyond the boundaries of elements becomes very complex--.

A simple alternative approach to overcome these problems is to store the entire text of XML documents in a single database attribute as a CLOB (Character Large Object), or a file outside the database.

In this paper we adopt the model-mapping approach for the following two reasons: a) the data structure of XML documents has more expressive power than the relational or object-oriented data model. This implies that we cannot find a method of structure mapping that maps data structures of XML documents into database schemas in a natural way. To cope with this problem, we need to extend the expressive power of database models [2]. However, storage schemes assuming extended database models are not applicable to off-the-shelf database systems; b) the structure mapping approach is suitable when we store a large number of XML documents that conform to a limited number of document structures or DTDs, and when the document structures or DTDs are static. However, numerous sophisticated Web applications are based on the flexible and dynamic usage of XML documents: (i) those whose DTDs are not known beforehand, or (ii) those that are well-formed but do not have DTDs. Further, many such applications deal with XML documents whose logical structure changes often. Obviously, the structure-mapping approach is inappropriate for storing a large number of such dynamic and structurally-variant XML documents.

Our main contribution is to propose a CORBA application to provide high available XML documents containing information about B2B transactions. The replication service is based on the use of a centralized replication manager and a set of geographically distributed repositories where documents are replicated. The replication manager stores for each document its template, its current version and its repository owner. The replication manager is in charge of the correct replication of the commercial transactions and its management, that is, the creation, modification and deletion of XML documents stored in the repositories geographically distributed. Each repository consists of an RDBMS and a local agent of the replication service, called Local Interface (LI), which translates XML documents to tables and vice versa using the model-mapping approach. While there is at least one active repository, the replication service ensures that users get the latest version of the document. It also supports failures at the local manager level and at the RDBMS. The manager replicates asynchronously the XML documents. Each operation over a document is always performed firstly in the repository owner of the document. Once it is done, the manager updates asynchronously all the replicas in the remainder repositories. Moreover, the manager follows a passive replication policy over all nodes. Whenever a document access fails, the manager receives an exception. This leads it to state which, by means of its local information, tries to recover the document from other repositories of the system. The system tolerates repository failures: LI and RDBMS stop failures.

The rest of the document is organized as follows: in section 2 the system model used in this replication service is
introduced; section 3 explains the communication and concurrency model used by our replication proposal; section 4 shows the replication model of our proposal; and finally conclusions end the paper.

2. SYSTEM MODEL

A B2B system generates an important amount of relevant information that must be appropriately managed. Because of the relevance of the information generated by the business logic of the B2B system, it is usually saved in a persistent storage system. From the user point of view, which is depicted in figure 1, users interact with the B2B system to perform a commercial transaction (CT). Users connect to web services to perform a CT. These services provide the interface between the user and the B2B system. Whenever a CT is performed, the B2B system saves it. In commercial systems, different kind of CT can be issued. CTs can be distinguished by the set of information they store about them. So, in an e-bookstore we have, for example, tracking CTs, that store user identity and the classification terms of the bought items; and selling CTs, that store the items the user has bought.

Figure 1. User’s vision of the system.

The information involved in each CT is very important to improve the quality of service of the system. Thus, this information is usually persistently stored by the B2B system. So, the B2B system requires storing the data of the CT. In this context, CTs are XML documents that are persistently stored. B2B systems manage information about CTs. Each one contains a set of information that constitutes its template. For example, a multipurpose template can be defined as the B2B exchange processor, the order transaction type, the name and port of the B2B exchange server, the billing transaction type, the login ID, the timeout, the Card Verification Value 2 (CVV2) configuration, the tax amount and the error, reject or success messages. Whenever a CT is performed, then a set of information corresponding to its template is persistently stored. In this paper, we propose the use of XML schemas to store both CT information and CT templates. Each template has associated a XML schema. The use of XML schemas allows to store properly the information and to recover it from the storage system when needed. In order to perform the storage of the XML documents generated in each CT, the system implements a storing service that allows to keep the different XML documents according to their templates. Commercial data and XML schemas are stored in a repository. Each one is composed by a RDBMS, where data is stored, and its local interface (LI). The LI is the interface between the web servers and RDBMS. It exchanges information with the web servers by means of CORBA, allowing applications to manage the commercial transaction documents. The LI receives storing/recovering requests of XML documents and translates locally these documents to a set of SQL sentences. LI demands to the RDBMS the execution of the SQL sentences in a transactional way, which allows the LI to store/recover the requested information. It inserts a new database register when storing information and provides an XML document for the web service when recovering the information. The above described elements are depicted in figure 2. At this point, we have a system that stores/recovers XML documents that users/applications can update using web services. In our case, the LI component provides the translation of XML to a set of SQL sentences using the Model-mapping approach. This proposal is similar to the one shown in [30].

However, the system is prone to fail-stop or crash errors. In order to improve the fault-tolerance of the B2B system we propose and implement a replication service (RS). The RS is necessary to ensure consistency when replicating repositories. The RS works in a replicate all policy. So, all non-faulty repositories eventually will contain the same set of information. Our RS is designed to ensure high availability of CT information by replicating repositories.

Figure 3 shows the system model after the introduction of multiple repositories and the replication service.

The RS is composed by two basic elements: the replication protocol (RP) and the information storage manager (ISM) that stores information processed by the protocol. The RS interacts with the ORB agents and manages the replicas throughout all the Internet, being this process totally transparent to the users. Each XML document has a translation to relational tables in an RDBMS. The representation of a document in an RDBMS is unique, although there are a set of templates according to the document nature. For example, a bank that offers online personal loans or banker’s orders, the kind of template used for the gener-
Establishes channel communications with the LIs of the system and detects failures in the replication system. The DSP protocol manages the storage of the XML documents into relational tables.

3. COMMUNICATION AND CONCURRENCY MODEL

Initially, the RS is listening, by means of the SCP, for the activation of any LI. Once one or more LIs are registered in the RS, they are selected to store CTs as XML documents. At predefined time intervals the RS requests to these LIs information about their state. The RS can obtain the state information by asking directly the LIs. It can also obtain it each time that LI ends an operation ordered by the RS, because the LI acknowledges the operation with a message containing its state information. These data are stored in the performance table managed by the ISM. From this information the RS is able to select the repository to store a new document to provide load balancing policies and, mainly, it is used to detect failures of LIs and RDBMS. These messages are exchanged by means of a reliable communication channel, since we are using CORBA as the mechanism of message passing which provides a free error communication channel.

Concurrency is established exclusively at RS level. When a new storage request arrives to the RS, a new execution thread is generated. This thread performs the replication task in the corresponding LI. Thus, the more requests appear, the more threads are initiated (one thread per request). Each LI has associated a queuing system where it introduces the storage requests. Each LI interacts with its RDBMS in a transactional way. All transactions share one table of the RDBMS (where metadata is stored), transactions are serialized to access this table, avoiding deadlock situations (deadlock free). The service is deadlock free.

4. REPLICATION MODEL

The ISM is in charge of information storing about the replication service. It stores: LI identifiers, their status (active or failed), the LI owner of each document, information about the document identifier including the time-stamp of the document update, and finally, references to the web servers owners of the documents. This is persistently stored in the local RDBMS of the RS. It is ordered by a performance ratio, in such a way that the LI located in the first place of the table is the candidate to the next storing operation because of its high performance ratio.

When an XML document is stored (the web servers interact with the RS through CORBA) the RS assigns the storage to the LI candidate (according to performance and proximity criteria). This LI is now the owner of the document.
The SCP connects with the LI owner, checks its state and if all is correct, the DSP sends the XML document to the LI owner. Then, the LI owner transforms the document in SQL sentences and the RDBMS executes them in a transactional way. Furthermore, a set of metadata containing the document identifier, the LI owner, and the document version is generated. Each XML document update implies the increase of the version number of the document. Once the RDBMS finishes correctly the transactions, it notifies the LI owner that all the process is performed without problems, and also its new load status. After that, the RS starts the replication of the document in the rest of active LIs. To do so, the SCP checks if all the LIs are working and the DSP sends the SQL sentences and the metadata to the LIs as it was explained earlier. During the replication process, the RS stores a temporal copy of this document until at least two repositories have a persistent copy of the document. This improves the document availability because we have at least two replicas of it in the system. Data replication is performed following the different cases now presented in table 1.

### New XML Document Creation


```sql
procedure create_xml;
begin
  choice LI_owner; /* performance table */
  send XML to LI_owner;
  transform XML to SQL_equivalent;
  save SQL_equivalent to RDBMStransactional;
  save Metadata to RDBMStransactional;
  if LI_owner = true then
    ∀ LI ≠ LI_owner do
      send (SQL_equivalent + Metadata)
  else fault_tolerance_algorithm
end;
```

### XML Document Update


```sql
procedure update_xml;
begin
  choice LI_owner; /* performance table */
  send XML_update to LI_owner;
  transform XML_update to SQL_equivalent;
  save SQL_equivalent to RDBMStransactional;
  save Metadata to RDBMStransactional;
  if LI_owner = true then
    ∀ LI ≠ LI_owner do
      send (SQL_equivalent + Metadata)
  else fault_tolerance_algorithm
end;
```

### XML Document Deletion


```sql
procedure delete_xml;
begin
  choice LI_owner; /* performance table */
  send XML_delete to LI_owner;
  transform XML_delete to SQL_equivalent;
  save SQL_equivalent to RDBMStransactional;
  if LI_owner = true then
    ∀ LI ≠ LI_owner do
      send (SQL_equivalent + Metadata)
  else fault_tolerance_algorithm
end;
```

4.1. Fault Tolerance

In order to ensure fault-tolerance and system recovering, an algorithm allowing to recover LIs, RDBMSs and the XML documents when they fall-down is introduced. Our proposal considers two kinds of failures: (i) LI and (ii) RDBMS stop fails. The replication manager recovers an LI failure by means of another repository where the replica has been stored. Note that eventually replicas will be stored in all repositories. In case of RDBMS fail, can be data lost or the RDBMS can be down, its content can be recovered by the RS from the replicas located in other repositories. Since LIs and RS are implementations of an Interface Definition Language (IDL) the RS can detect an LI failure if it raises an exception. We consider that an LI has crashed when, after calling an operation on it, it raises an exception. We can define different cases involved in the recovery process (see table 2 where the recovery algorithm is depicted):

- An LI has crashed (LI status = failed) and restarts (LI status = active). The recovering task starts by asking the ISM for the sites containing the documents stored in the damaged LI. For each one of them, the RS compares document versions by means of metadata. If the document version of any document stored in the rest of LIs is newer than the stored one in the damaged LI, then the RS orders to transfer the documents with the newer versions to the LI. The RS uses SQL and metadata information to perform this task.

- Documents stored in the system during the inoperative time interval of the damage LI are stored in the rest of LIs, but not in the damaged one. So, a recovering process is needed in order to store in the LI these documents. Then the RS locates the newer versions of the mentioned documents and copies them (document and metadata associated) from the LIs owners to the damaged LI.

- Same as the previous item, when documents are deleted during the inoperative time interval of the damage LI, the RS ensures an appropriate deletion of the document and metadata copies in the damaged LI.

### Table 1. Data Replication Process.
// Status(L1) changes from failed to active.
var maxversion : integer;

/* case 1 */
∀ D_i ∈ L1_i do
begin
    process1(D_i);
    if ∃ D_j and D_i = D_j then
    begin
        process2(D_j);
        if version(D_i) = version(D_j) then
        begin
            maxversion := version(D_j);
            ∀ L1_i ≠ L1_j and (status(L1_j) = active) do
            begin
                process1(D_j);
                if maxversion = version(D_j) then
                begin
                    maxversion := version(D_j);
                end
            end
        end
    end;
end;

/* case 2 */
∀ D_i ∈ L1_i do
begin
    process1(D_i);
    if not ∃ D_j and D_i = D_j then
    begin
        process2(D_i);
        if version(D_i) = version(D_j) then
        begin
            maxversion := version(D_j);
            ∀ L1_i ≠ L1_j and status(L1_j) = active do
            begin
                process1(D_j);
                if maxversion = version(D_j) then
                begin
                    maxversion := version(D_j);
                end
            end
        end
    end;
end;

/* case 3 */
∀ D_i ∈ L1_i do
begin
    process1(D_i);
    if ∃ D_j and D_i ≠ D_j then
    begin
        process3(D_j);
    end;
end;

procedure process1(D);
begin
    send_recovery(idD) to L1;
    receive_data(idD) to RDBMS, transactional
end;

procedure process2(maxversion, L1_i);
begin
    send_recovery(idD) to L1;
    receive_data(idD) to RDBMS, transactional;
    send(SQL_equivalent + Metadata) to L1_i;
    save(SQL_equivalent + Metadata) to RDBMS, transactional
end;

procedure process3(D);
begin
    send(SQL_equivalent + Metadata) to RDBMS, transactional
end;

Table 2. The Recovery algorithm.

5. Conclusions

This paper introduces an on demand replication service for B2B transactions, that is one of the major concerns of the Internet today, stored as XML documents.

The architecture consists of a group of web servers that are connected to a replication service. These web servers utilize this service to store and recover commercial transactions performed by web clients whenever anyone of them crashes. The replication architecture consists in a group of nodes and a replication manager interconnected by means of CORBA, which ensures reliable connections among them, avoiding node grouping and multicast among the replication servers. The system replication task, which consists in an asynchronous passive replication to all node replicas, under normal conditions is depicted in this work. It is also outlined the way a node recovers after its failure, this process is handled by the replication manager itself.

XML documents are persistently stored in the replicated servers by means of an RDBMS. The replication server makes use of the mapping-model approach that ensures a higher performance in the storage and recovery process of XML documents stored in an RDBMS.

Acknowledgements

This work has been partially supported by the Spanish Government under research grant CICYT TIC2003-09420-CO2-02.

References


