International Journal of Reviews in Computing

IJRIC

www.ijric.org E-ISSN: 2076-3336

DESIGN OF HETEROGENEOUS DATABASES REPLICATION USING XML

[1,2]SEIFEDINE KADRY, [1,2]HUSSAM KASSEM, [2]MOHAMAD SMAILI

[1]Lebanese University - CNAM, Lebanon

[2]Lebanese University - Faculty of Science, Lebanon

E-mail: skadry@gmail.com, hussamkassem@yahoo.com, mosmaili@ul.edu.lb

ABSTRACT

Nowadays, the application software and the variety of the (DBMS) are in increasing expansion, and the replications between different database systems due to different infrastructures are more than needed. Since, the replication of data between different DBMS is not always possible. We propose in this paper, a new design to solve the replication problem between heterogeneous DBMS systems using XML technology.

Keywords: *Database Management Systems (DBMS), Replication, eXtensible Markup Language (XML), Structured Query Language (SQL) Server, Oracle.*

1. INTRODUCTION

Presently, most companies have different systems, i.e. payroll system, financial system, audit system, core system, etc... These systems use different database management systems (e.g. oracle, Sybase, SQL Server, DB2,…), they should communicate and some times may need to replicate to generate a unified report from one system. The replication between these databases is very difficult due to the internally heterogeneous structure.

In this paper, a proposed design to solve this problem is given. This design uses XML as Middleware and .NET as interface.

The paper is organized as follows. Section 2 describes the replication, the replication software, the replication types, and the replication metaphors. In Section 3, we present the XML language, and the relation with DBMS. The proposed design is given in section 4. Finally we draw the conclusions.

2. REPLICATION

Replication is a "set of technologies" that can move and copy data and database objects from one DBMS to another of the same type or different, across different platforms and geographic locales. This allows users to work with a local copy of the database, and any changes made are transferred to one or more remote servers or mobile users across the network.

The replication is a synonym of distributed data. Furthermore, it replicates data to multiple database copies, and the consistency of the database is maintained by the process of synchronization. The question is how to distribute data using replication, when, where, and how data is propagated.

In the following subsections, we define the replication software, the replication types and the replication metaphors.

2.1 REPLICATION SOFTWARE

Each DBMS has its own module (i.e. replication between SQL Server and SQL Server, Oracle and Oracle…), but we don't have replication software between different DBMSs (i.e. between DB2 and Oracle, SQL Server and Sybase…).

2.2 REPLICATION TYPES

In general, we have two types of replication: Eager and Lazy replications [1].

2.2.1 EAGER REPLICATION

Eager replication is also known as synchronous replication. In this method, an application can update a local replica of a table, and within the same transaction, it can also update other replicas of the same table. No concurrency anomalies occur, since synchronous replication gives serializable execution. Any anomaly in concurrency is detected by the locking method. If any of the nodes are

disconnected, eager replication prevents the update from taking place. Furthermore, the atomicity of the transactions is guaranteed by the employment of the 2PC method. However, there is a compromise in performance as a result of all the updates being carried in a single transaction.

Eager replication consists of the following steps: execute, transmit, notify, and either commit or rollback. An executed transaction is transmitted to different nodes, and in the event of failure in one node, the transaction is rolled back and all the other nodes are notified of the failure. The transaction is then aborted in all nodes. If replication is successful in all the nodes, a commit is broadcast and a copy of the committed transaction is then sent to all the nodes. This type of replication is illustrated in Figure 1.

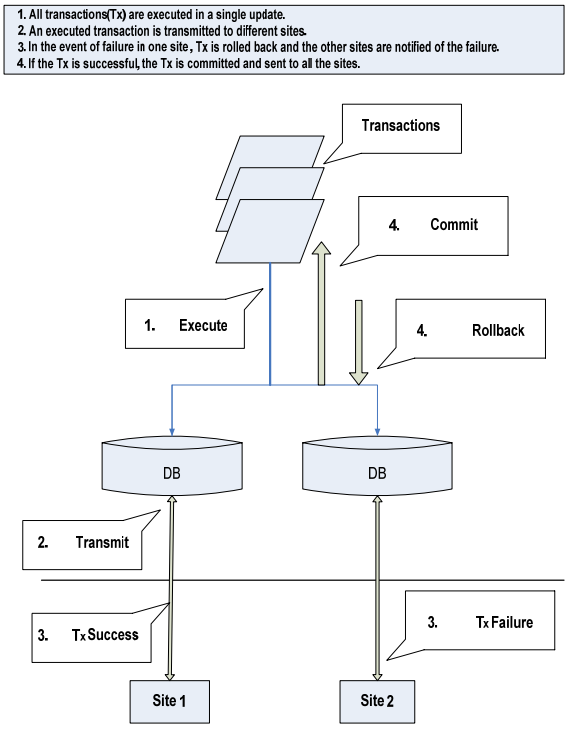


Figure 1: Eager Replication

2.2 LAZY REPLICATION

Lazy replication is also known as asynchronous replication. In this case, if the transactions are committed, they are sent to the different sites for the updates to occur. However, if they are rolled back, the changes will not be transmitted to the different sites. Thus, the very nature of asynchronous replication allows the updates of committed transactions to be sent to disconnected sites, as in the case of handheld sets or mobile devices. This is shown in Figure 2.

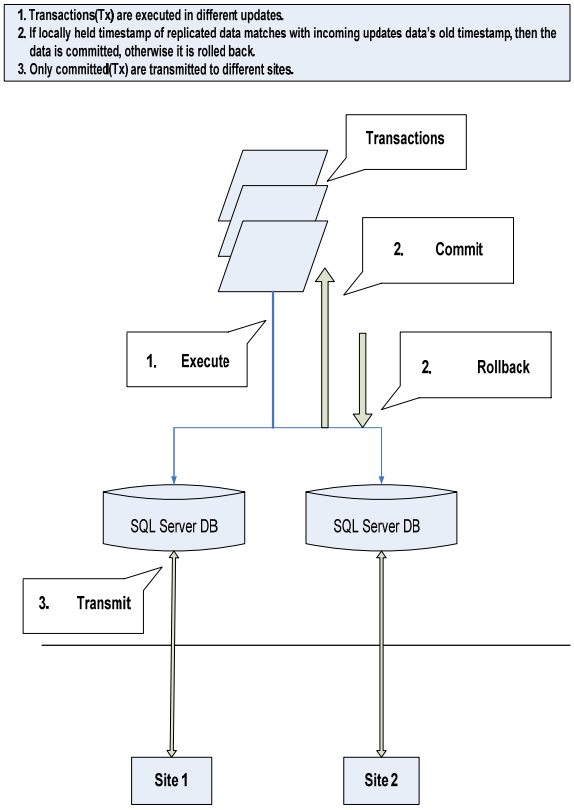


Figure 2: Lazy Replication

With this type of replication, it is possible for two different sites to update the same data on the same destination site. This will lead to a conflict in the updating of the data. Such update conflicts need to be resolved in lazy replication, and this is done by associating timestamps with each of the transaction objects. Each object carries the timestamp associated with the previous update of that data. So when a transaction is sent to the destination site, it first checks to see whether the timestamp associated with the local copy of the replicated data matches the incoming transaction's old timestamp for that data. If they match the changes, including the transaction's new timestamp, the updated transaction will be applied. If the timestamps do not match at the initial stage, the updated transaction is rejected. SQL Server, for example, has a conflict resolution viewer that deals with updates, inserts, and deletes.

2.3 REPLICATION METAPHORS [1]

www.ijric.org E-ISSN: 2076-3336

Publisher

The publication server (or publisher) contains the database or databases that are going to be published. This is the source of the data (replicated) to other servers.

Distributor

The distribution server (or distributor) can either be on the same server as the publication server or on a different server (in which case it is a remote distribution server). This server contains the distribution database. This database, also called the store-and-forward database, holds all the data changes that are to be forwarded from the published database to any subscription servers that subscribe to the data. A single distribution server can support several publication servers. The distribution server is truly the workhorse of data replication.

Subscriber

The subscription server (or subscriber) contains a copy of the database or portions of the database that are being published. This is known as store-and-forward.

Article

An article is any grouping of data to be replicated; it is a component of a publication. It may contain a set of tables or a subset of tables. Articles can also contain a set of columns (vertical filtering), a set of rows (horizontal filtering), stored procedures, views, indexed views, or User Defined Functions (UDFs).

Publication

The Publisher server contains a collection of articles in the publication database. This database tells the Publisher server which data needs to be sent to other servers or to the subscribing servers. In other words, the publication database acts as the data source for replication.

Any database used as a source of replication needs to be enabled as a Publisher server. The published database contains one or more publications. A publication is a unit which contains one or more articles that are sent to the subscribing servers.

Subscriptions

Subscriber servers must define their subscriptions for a particular set of publications in order to receive the snapshot from the publisher server.

Agents

They are the processes responsible for copying and distributing data between publisher and subscriber. There are different types of agents supporting different types of replications

3. XML

3.1 XML ORIGIN AND GOALS [2], [3]

XML stands for eXtensible Markup Language defined by World Wide Web Consortium (W3C- www.w3c.org). Markup languages describe the form of the document; this form is the way of describing how the content of the document should be interpreted.

Therefore, the word Markup here is misleading; XML ought to refer eXtensible Meta Language, because it is a standardized, flexible, following precise, exacting rules for structure, syntax, and semantics of data to be interpreted. XML is not a programming language, it is a grammar used to define and describe data structures. XML is popular for many reasons, among:

Easy Data Exchange

In earlier days, programs could exchange data easily because data was stored as text. But, today, we need conversion programs or modules to let applications transfer data between themselves. In fact, proprietary data formats have become so complex that frequently one version of an application can't even read data from an earlier version of the same application.

Customizing Markup Languages

You can create customized markup languages using XML, and that represents its extraordinary power.

Self-Describing Data

The data in XML documents is self-describing. Example:

```
<?xml version="1.0"?>
<greeting style="informal">
  <from>Chris Bates</from>
  <to>Mr. M. Mouse</to>
  <message>Hi, how are you doing?</message>
  <signature />
</greeting>
```

Based solely on the names given to each XML element here, we can figure out what's going on. Each element describes itself, this means that XML documents are, to a large extent, self-documenting.

Structured and Integrated Data

XML can specify the data, the structure, and how various elements are integrated into other elements. This is important when we are dealing with complex and important data. In XML, we can

www.ijric.org E-ISSN: 2076-3336

represent a long bank of statement, and build in the semantic rules that specify the structure of the document. So that, the document will be checked correctly.

Let us read the XML document:

```
<apartmentbldg>
  <apartment>
   <room type="bedroom">
    <furniture type="armoire">
     <belonging type="t-shirt" color="navy" size="xl" />
     <belonging type="sock" color="white" />
     <belonging type="watch" />
    </furniture>
   </room>
  </apartment>
</apartmentbldg>
```

If we check the code, we will notice that the different elements are nested according to their physical locations within the building.

In fact, this emphasis on the correctness of documents is strong in XML. since, browsers are supposed to check our document; if there's a problem, they are not supposed to proceed any further.

3.2 WELL-FORMED AND VALID XML DOCUMENTS

To be well-formed, an XML document must follow the syntax rules set up for XML by W3C in the XML 1.0 specification (which we can find at www.w3.org/TR/REC-xml).

Informally, well-formedness often means that the document must contain one or more elements. The root element, must contain all the other elements. Each element also must nest inside any enclosing elements properly. For example, this document is not well-formed because the </GREETING> closing tag comes after the opening <MESSAGE> tag for the next element:

```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT>
  <GREETING>
    Hello From XML
  <MESSAGE>
  </GREETING>
    Welcome to the wild and woolly world of XML.
  </MESSAGE>
</DOCUMENT>
```

An XML document is valid if there is a Document Type Definition (DTD) associated, and if the document complies with that DTD.

A document's DTD specifies the correct syntax of the document, DTDs can be stored in a separate file or in the document itself, using a <!DOCTYPE> element. Here's an example in which a <!DOCTYPE> element is added to the greeting XML document developed previously:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <?xml-stylesheet type="text/css" href="first.css"?>
3. <!DOCTYPE DOCUMENT [
4. <!ELEMENT DOCUMENT (GREETING, MESSAGE)>
5. <!ELEMENT GREETING (#PCDATA)>
6. <!ELEMENT MESSAGE (#PCDATA)>
7. ]>
8. <DOCUMENT>
9. <GREETING>
10. Hello From XML
11. </GREETING>
12. <MESSAGE>
13. Welcome to the wild and woolly world of XML.
14. </MESSAGE>
15 </DOCUMENT>
```

Let us examine the above document:

- Line 3 declares an inline DTD called document and that anything inside the square brackets [] comprises the inline DTD. In this case, Document is the root element.
- Line 4 indicates that the root element contains Greeting element followed by Message element. Therefore, an XML instance with a Message element before Greeting element would not be valid according to this DTD.
- Line 5 and 6 declare that Greeting and Message element contains only text data, we could validate this in DTD by specifying the type PCDATA (Parsed Character DATA).

Most XML browsers check XML documents for well-formedness, but, few of them check for validity. Here's a XML validator on the Web:

W3C XML Validator, validator.w3.org/. This is the official W3C HTML validator. Although it's officially for HTML, it also includes some XML

support. The XML document must be online to be checked with this validator.

P.S: A valid document is always a well-formed document, but the reverse is not always true.

3.3 XML AND RDBMS [4], [5]

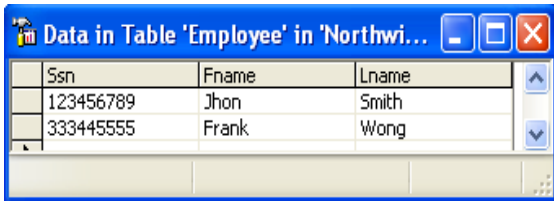
This section focus on the relation between XML and RDBMS. To simplify and clarify this relation, two cases studies using SQL Server 2005 and Oracle 10g databases are illustrated.

Retrieving or **selecting** data using XML appears as XML documents after executing a SELECT query. Also **updating, inserting, deleting** can be done using XML.

3.3.1 SQL SERVER AND XML

In the following, two examples are given. The first one is select data from an SQL server table ("Employee") into an XML file. The second is insert data into the table form an XML file.

Select statement using FOR XML clause



To output the result of the select statement as XML format, we use the FOR XML clause PATH mode:

```
SELECT Ssn AS
"Employee/@SocialSecurityNumber",
Fname AS "Employee/FirstName",
Lname AS "Employee/LastName"
FROM dbo.EMPLOYEE
FOR XML PATH,
ELEMENTS XSINIL;
```

FOR XML PATH is designed for explicitly defining your XML result structure. This is a safer option than the RAW or AUTO modes in production code because, we always know the result XML structure in advance, even if the table structure changes.

The @ element in the first row is to indicate that is considered as an attribute. The XSINIL modifier tells FOR XML to represent SQL NULLs with an xsinil = "true" attribute in the resultant XML.

The resultant XML:

```
<row
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
 <Employee SocialSecurityNumber="123456789">
  <FirstName>Jhon</FirstName>
  <LastName>Smith</LastName>
 </Employee>
</row>
<row
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance">
 <Employee SocialSecurityNumber="333445555">
  <FirstName>Frank</FirstName>
  <LastName>Wong</LastName>
 </Employee>
</row>
..
..
..
```

Inserting data using OPENXML

OPENXML primarily used to insert data from an XML document into tables in the database. Before using the OpenXML function to insert data, the XML document needs to be **parsed** and mapped to an in-memory tree structure that represents the nodes in the document. We use the ***sp_xml_preparedocument*** stored procedure, which reads the document and verifies that it's a valid XML document. The stored procedure then returns a handle to a node tree that can be used to retrieve data from the elements and attributes in the document.

After the node tree has been created, you can use the OpenXML function to return a rowset containing data in the XML document. The primary use of this functionality is to get XML data into a relational format so that it can be inserted into a table. This process is known as **shredding** the document.

Once a document has been fully processed, you should use the ***sp_xml_removedocument*** stored procedure to reclaim the memory used by the node tree.

Example: following, the steps to insert into the table "Department" a new department with 2 employees using ***openxml***.

The XML document used:

```
<?xml version="1.0"?>
```

www.ijric.org E-ISSN: 2076-3336

```
<Department DepartmentNumber="20"
DepartmentName="Information Technology"
  ManagerSocialSecurityNumber="999887777"
ManagerStartDate="10/20/1905">
  <Employees>
   <Employee
SocialSecurityNumber="765765765">
      <FirstName>Hussam</FirstName>
      <MiddleName>N</MiddleName>
      <LastName>Kassem</LastName>
      <BirthDate>04/20/1971</BirthDate>
      <Address>Bekaa,Lebanon,Tamnin Fawka
str</Address>
      <Sex>M</Sex>
      <Salary>30000.00</Salary>

<SuperSocialSecurityNumber>999887777</SuperS
ocialSecurityNumber>
    </Employee>
    <Employee
SocialSecurityNumber="222111333">
      <FirstName>Seifedine</FirstName>
      <MiddleName>N</MiddleName>
      <LastName>Kadry</LastName>
      <BirthDate>07/15/1977</BirthDate>
      <Address>Bekaa,Lebanon,Rafid
str</Address>
      <Sex>M</Sex>
      <Salary>40000.00</Salary>

<SuperSocialSecurityNumber>999887777</SuperS
ocialSecurityNumber>
    </Employee>
  </Employees>
</Department>
```

The tree structure of the above XML document is:

- /(root)
  - Department(@DepartmentNumber,@DepartmentName,@ManagerSocialSecurityNumber,@ ManagerStartDate)
    - Employees
      - Employee(@SocialSecurityNumber)
        - FirstName
        - MiddleName
        - LastName
        - BirthDate
        - Address
        - Sex
        - Salary
        - SuperSocialSecurityNumber
      - Employee(@SocialSecurityNumber)
        - FirstName
        - MiddleName
        - LastName
        - BirthDate
        - Address
        - Sex
        - Salary
        - SuperSocialSecurityNumber

The following procedure inserts the xml data into the "Department" table:

```
CREATE PROCEDURE InsertDepartmentEmployees
@xmlDepEmp VARCHAR(2000)
AS
DECLARE @iTree INTEGER
EXEC sp_xml_preparedocument @iTree OUTPUT,
@xmlDepEmp
INSERT dbo.DEPARTMENT
(Dnumber,Dname,Mgr_ssn,Mgr_start_date)
select
DepartmentNumber,DepartmentName,ManagerSo
cialSecurityNumber,ManagerStartDate
FROM
OPENXML(@iTree, 'Department', 1)
WITH (DepartmentNumber INTEGER,
DepartmentName VARCHAR(15),
ManagerSocialSecurityNumber
CHAR(9),ManagerStartDate DATETIME)
INSERT dbo.EMPLOYEE
select * FROM
OPENXML(@iTree,'Department/Employees/Emplo
yee',1)
WITH (SocialSecurityNumber CHAR(9),FirstName
VARCHAR(15) 'FirstName',MiddleName CHAR(1)
'MiddleName', LastName VARCHAR(15)
'LastName', BirthDate DATETIME
'BirthDate',Address VARCHAR(30) 'Address', Sex
CHAR(1) 'Sex',Salary DECIMAL 'Salary',
SuperSocialSecurityNumber CHAR(9)
'SuperSocialSecurityNumber',
    DepartmentNumber INTEGER
'../../@DepartmentNumber')
EXEC sp_xml_removedocument @iTree;
```

3.3.2 ORACLE AND XML [6], [7]

As we discussed in the previous section the relation between SQL Server and XML, we discuss in this section the relation between Oracle and XML. Oracle offers more support for XML than SQL Server 2005.

The ***Oracle XML Database (XML DB)*** refers to the collection of XML technologies built into the

www.ijric.org E-ISSN: 2076-3336

Oracle 10g database, providing high-performance and native storage, retrieval, and processing of XML.

Select data from Oracle table into XML file

The below query selects data from the table employee and outputs the result in XML format:

```
SELECT XMLELEMENT("Employee",
XMLATTRIBUTES ( e.fname ||' '|| e.lname AS
"name" ),
XMLFOREST( e.Bdate as "BirthDate", e.Address))
AS "result"
FROM cv2.employee e;
```

Where:

- **xmlelement()** function creates the root element. "Employee".
- **xmlattibutes()** may be used only as an argument of **xmlelement()**.
- **xmlforest()** produces a forest of XML elements from the given list of arguments.

Following the XML output of the previous query:

```
<Employee name="Jhon Smith">
 <BirthDate>09/01/65</BirthDate>
 <ADDRESS>731 fondron,houston,tx</ADDRESS>
</Employee>
<Employee name="Frank Wong">
 <BirthDate>08/12/55</BirthDate>
 <ADDRESS>638 voss,houston,tx</ADDRESS>
</Employee>
<Employee name="James Borg">
 <BirthDate>10/11/37</BirthDate>
 <ADDRESS>450 stone,houston,tx</ADDRESS>
</Employee>
<Employee name="Alicia Zelaya">
 <BirthDate>19/01/68</BirthDate>
 <ADDRESS>3321,castle,spring,tx</ADDRESS>
</Employee>
<Employee name="Jenifer Walace">
 <BirthDate>20/06/41</BirthDate>
 <ADDRESS>291 berry,bellaire,tx</ADDRESS>
</Employee>
<Employee name="Ramesh Narayan">
 <BirthDate>15/09/62</BirthDate>
 <ADDRESS>975 fireoak,humble,tx</ADDRESS>
</Employee>
<Employee name="Joyce English">
 <BirthDate>31/07/72</BirthDate>
 <ADDRESS>5631 rice,houston,tx</ADDRESS>
</Employee>
<Employee name="Ahmad Jabbar">
 <BirthDate>29/03/69</BirthDate>
 <ADDRESS>980 dallas,houston,tx</ADDRESS>
</Employee>
```

Inserting data to department table from XML file

Inserting into "cv2.department" table: DATE type inserted as DAY/ MONTH /YEAR. unstructured inserting method using CLOB.

```
DECLARE
insCtx DBMS_XMLSTORE.ctxType;
rows NUMBER;
xmlDoc CLOB :=
'<ROWSET>
 <ROW num="1">
  <DNAME>IT</DNAME>
  <DNUMBER>20</DNUMBER>
  <MGR_SSN>123456789</MGR_SSN>

<MGR_START_DATE>23/12/2000</MGR_START_D
ATE>
 </ROW>
 <ROW>
  <DNAME>communication</DNAME>
  <DNUMBER>30</DNUMBER>
  <MGR_SSN>123456789</MGR_SSN>

<MGR_START_DATE>20/11/1995</MGR_START_D
ATE>
 </ROW>
</ROWSET>';
BEGIN
insCtx :=
DBMS_XMLSTORE.newContext('cv2.Department');
-- Get saved context
DBMS_XMLSTORE.clearUpdateColumnList(insCtx);
-- Clear the update settings
-- Set the columns to be updated as a list of values
DBMS_XMLSTORE.setUpdateColumn(insCtx,
'DNAME');
DBMS_XMLSTORE.setUpdateColumn(insCtx,
'DNUMBER');
DBMS_XMLSTORE.setUpdateColumn(insCtx,
'MGR_SSN');
DBMS_XMLSTORE.setUpdateColumn(insCtx,
'MGR_START_DATE');
-- Insert the doc.
rows := DBMS_XMLSTORE.insertXML(insCtx,
xmlDoc);
DBMS_OUTPUT.put_line(rows || ' rows inserted.');
-- Close the context
DBMS_XMLSTORE.closeContext(insCtx);
END;
```

www.ijric.org E-ISSN: 2076-3336

4. THE PROPOSED DESIGN

In the section, the restate and the illustration of the problem are given. In addition, the completed proposed design is drawn.

4.1 PROBLEMATIC

Today, most of the business companies have more than one software to manage the whole business process. For example, we find in the Bank one software for the core business (accounts, transactions, balance,…), one software for the Human Resources department (employees records, salaries, bonus,…), and another software for the Private Banking department (Bonds, treasury, …) …etc.

We always need to generate a unified report between these different systems which use different database management systems (DB2, Oracle, Sybase, SQL Server…). The replication is one of the major solutions to generate this report and avoid the import/export error in exchange data between different systems. We do often face problems in the replication scenario due to the heterogeneous (i.e., internally different) database systems structures.

4.2 SUGGESTED DESIGN

Our proposed design to solve the replication problem is based on the XML technologies. As we mentioned previously, the XML is the most appropriate language for data exchange. Furthermore, most of the databases today support for XML, which means we can create an XML file or read from XML file easily. Therefore, our new idea is to use the XML as middleware between different databases, and .NET as interface.

The XML solution for replicating data between, for example, SQL Server and Oracle, is achieved if it provides:

1- Solution for Offline Backup Replication.
2- Solution for Online Backup Replication.
3- Data Consistency, Concurrency, Recovery Solution.

4.2.1 SOLUTION FOR OFFLINE BACKUP REPLICATION

A row or an entire table, saved as XML file, it will be replicated **asynchronously** as a backup operation to a database server. The offline row replication will assist to check the data consistency.

Each database server has access to its offline XML files from a specific shared replication folders, in other words, if we replicate data among three databases, each database has two shared folders, one folder holding XML files sent from a database server, another folder holding XML files sent from the other database server (*cf.* figure 3).

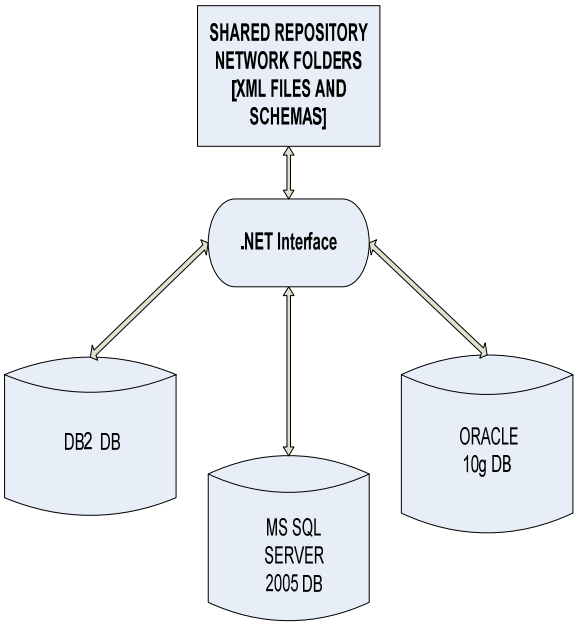


Figure 3: Offline Replication

In figure 3, the folders of SQL Server offline XML files, hold the files sent by Oracle **(OracleToSQL)**, and DB2 **(DB2ToSQL)**.

Each replication folder contains:

Offline Row replication files (XML files) for inserting, deleting, updating, are saved in the shared replication folder according to the following **names convention:**

The replication type (offline), the database server sending the file, the kind of operation (insert, delete, update) followed by the table name, and a serial number of nine digits (generated systematically when the data is serialized as XML)

Example for inserts:
Off_Ora_Ins_employee220320090.xml,
Off_Ora_Ins_employee220320091.xml, ……

Off: means offline replication, **Ora:** sent from Oracle, **Ins:** insert file number 220320090 in **employee** table.

The number 220320090 is read as follows: the first eight digits 22032009 represents a date casted

to a long integer number, the last digit represents a file serial number.

***Example for updates*:**
Off_Ora_Upd_employee220320090.xml,
Off_Ora_Upd_employee220320091.xml,.....

These files are saved daily for backups.

Offline replication file per database **table**: (replicated once per day)

Note that the online row replication takes priority over offline.

Ex: Off_Ora_employee22032009.xml, Off_Ora_departemnt22032009.xml, ….

4.2.2 SOLUTION FOR ONLINE BACKUP REPLICATION

In online backup replication, a row is triggered for each DML (insert, update, delete) query, as XML document in a repository folder, and a process **(JOB)** from other server side check the changes to upload the data **synchronously**.

For any DML task (insert, delete, update) in any database, a trigger is fired to allow an XML document containing the DML task to be saved in the shared repository folder. A process running on each database servers, tracks the changes, and the XML data is uploaded to the databases (*cf.* figure 4).

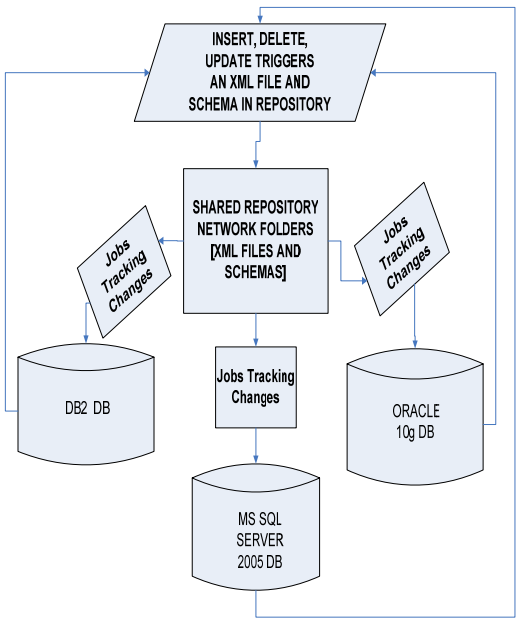


Figure 4: Online Replication

For each new row added, deleted, or updated, the row is triggered as an XML document from **'Inserted'** in sender instance server, saved in a shared repository folder, and uploaded to correspondent table in receiver instance server.

Our design concerns data replication between different databases systems, where the database management is managed in each database system separately from the other ones (which is known by the autonomous database management).

There are some requirements:

- The local operations (queries process, transaction management, system administration…) of the DBMSs must be not affected by their participation in multidatabase replication process.
- The system consistency or operation should not be compromised when DBMS involved in data replication task.
- The DBMSs are free to use the data models and transaction management techniques without interfering with each other (design autonomy).
- Each of the DBMSs is free to make its own decision as to what type of information it wants to provide to the other DBMSs (communication autonomy).

The level of communication among DBMS systems in our design is the process of data replication as XML documents.

For a replication task between SQL Server and Oracle, the data consistency is maintained programmatically as follows:

a- From SQL Server to Oracle

From SQL Server database, an XML document containing data that will be replicated to Oracle database, is saved in a **shared repository folder (MSSQLServer ToOracle)** accessed by Oracle database. **(SQL Server: Write, Oracle: Read and replicate),** (*cf.* figure 5). This folder contains XML documents, and in each table in the database, there are:

Three XML documents for online replication, named by current date convention. As an example, the **'cv2.employee'** table in **'cv2.schema'** in Oracle has:

- Document for online insert replication with its corresponding lock file.

www.ijric.org E-ISSN: 2076-3336

(ex: file name: On_SQL_Ins_employee-22-05-2009.xml)&(SQLOraLockIns.txt).

- Document for online update replication with its corresponding lock file.

(ex: file name: On_SQL_Upd_employee-22-05-2009.xml)&(SQLOraLockUpd.txt).

- Document for online delete replication with its corresponding lock file.

(ex: file name: On_SQL_Del_employee-22-05-2009.xml)&(SQLOraLockDel.txt).

On: Online, SQL: sent from SQL Server, Ins: insert, Upd: update, Del: delete.

The Online (insert, update, delete) replications, which is synchronous, in Oracle are done for each data row manipulated in the original SQL Server database tables.

For each online operation, the new online data are inserted into the empty XML file (old data in the XML file are deleted by Oracle job sequence tasks).

The XML documents names in the shared folder are changed daily according to the current date convention.

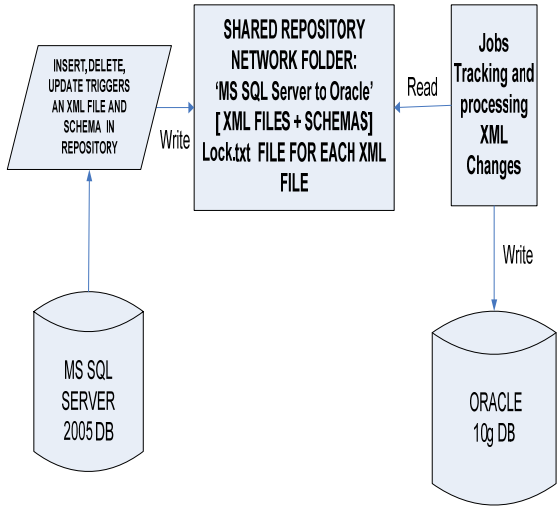


Figure 5: Shared XMLs Folder (SQL Server to Oracle)

b- From Oracle to MS SQL Server

From Oracle database, an XML document containing data will be replicated to SQL Server database, is saved in a **shared repository folder (OracleToMSSQL Server)** accessed by the SQL Server database. **(Oracle: Write, SQL Server: Read and replicate), (cf.** figure 6). This folder contains XML documents, and in each table in the database, there are:

Three XML documents for online replication, and one for offline, named by current date convention. As an example, the **'employee'** table in **'employeedb'** in SQL Server database has:

- Document for online insert replication with its corresponding lock file.

(ex: file name: On_Ora_Ins_employee-22-05-2009.xml)&(OraSQLLockIns.txt).

- Document for online update replication with its corresponding lock file.

(ex: file name: On_Ora_Upd_employee-22-05-2009.xml)&(OraSQLLockUpd.txt).

- Document for online delete replication with its corresponding lock file.

(ex: file name: On_Ora_Del_employee-22-05-2009.xml)&(OraSQLLockDel.txt).

Online (insert, update, delete) replications, which is synchronous, in SQL Server are done for each data row manipulation in the original Oracle database tables.

For each online operation, the new online data are inserted into the empty XML file (old data in the XML file are deleted by SQL Server job sequence tasks).

The XML documents names in the shared folder are changed daily, and named according to the current date convention.

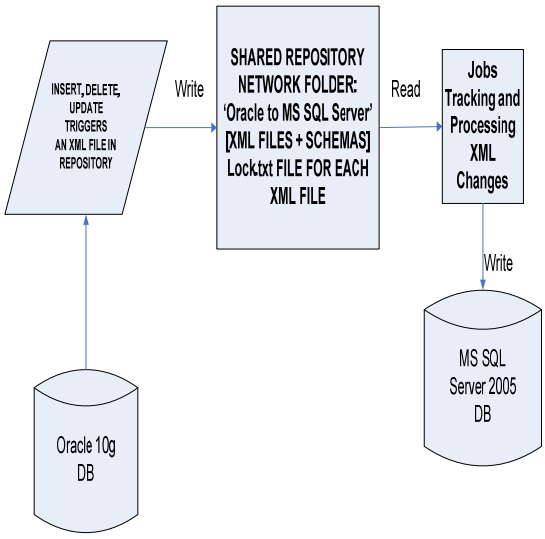


Figure 6: Shared XMLs Folder (Oracle to SQL Server)

www.ijric.org E-ISSN: 2076-3336

4.2.3 LOGICAL FLOW OF ONLINE REPLICATION TASK

a- SQL Server - XML Logic Flow

The **Common Language RuntimeTrigger** (CLR) is the heart and soul of the Microsoft .NET Framework. It provides the environment for the execution of all the .NET Framework code. By integrating the CLR into SQL Server, developers can now write stored procedures and other objects, compile them into managed code (code that runs within the CLR and compiled to .dll as assembly), and use them right from SQL Server [5], [8].

Figure 7 shows the logical flow of loading SQL Server DML insert, update, and delete operation to XML file.

The flag variable in Lock.txt solves the problem of concurrency, in which it avoids Oracle job from reading, writing then deleting XML data, while SQL Server is loading the DML operation to XML file. It avoids also SQL Server from loading a DML operation to XML file, while Oracle job is reading, writing then deleting the XML data file.

b- XML - Oracle Flow

Figure 8 shows the logical flow of reading and writing the loaded SQL Server XML file to Oracle database and log table. You can use the **Job type: ODP.NET PL/SQL block** [9].

4.2.4 DATA CONSISTENCY, CONCURRENCY, RECOVERY SOLUTION

Data consistency

Based on log files technique, we create a **log table** for each database server table concerned in the replication tasks:

- On SQL Server, we may have log tables for **'employee' (LogSQLemployee)**, for **'department' (LogSQLdepartment)**, etc…….
- On Oracle **(LogOracleemployee, LogOracledepartment,**…..**)**.

The idea is to use these tables for **Reconciliation** (data existence on both database servers), by adding to the **job** that will read XML document from repository, a piece of code to write the changes to the log table after writing them to the original database table, and later we can compare the log table with the original one.

Data Concurrency

Concurrency problem arises when both database servers (i.e. Oracle and SQL server) access the same XML document and one of them (or both) is writing on to it. The server reading the XML document, might read the unchanged XML document. This problem is avoided by using separated XML files on each database servers, in addition to the flag variable in **'Lock.txt'** that locks the file being accessed by a server.

Data Recovery

The system fails if the network connection, or the online row replication fail. Therefore, the XML documents become damaged or inaccessible, we can switch to the offline replication - waiting for system or network connection recovery— by replicating the entire table and recreate the repository online files to rework again with online replication.

5. CONCLUSION AND FUTURE WORK

This paper introduces a new design to solve the replication problem between heterogeneous DBMS systems using XML technology as Middleware. The replication is very difficult due to the internally heterogeneous structure. We are working on implementing our design. Moreover, we may design the .NET applications for Oracle, and new database systems, and we can implement the **online** .NET applications from SQL Server to Oracle replication, from SQL Server to XML, and from XML to Oracle.

www.ijric.org E-ISSN: 2076-3336

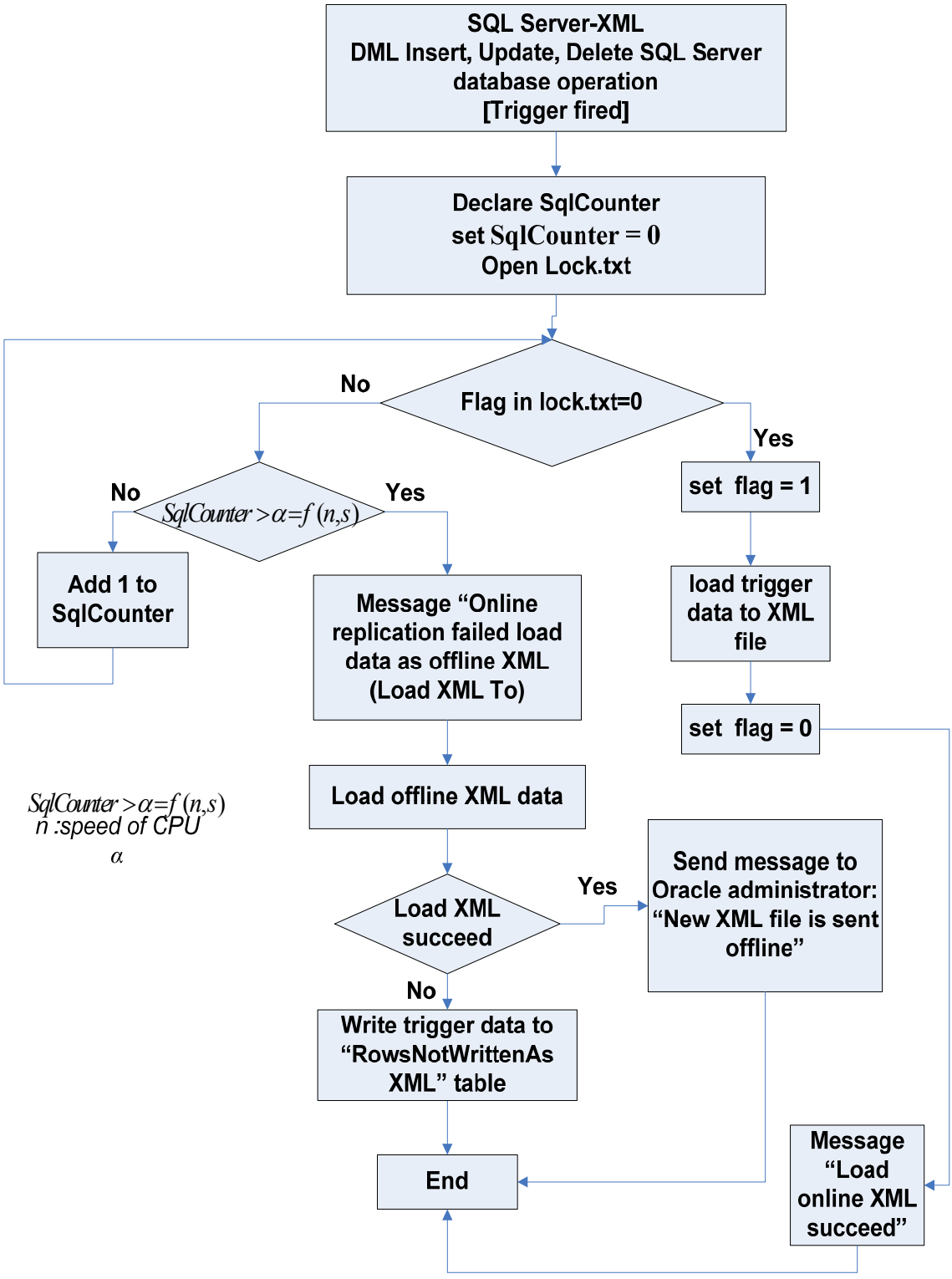


Figure 7: SQL Server - XML Logic Flow

www.ijric.org E-ISSN: 2076-3336

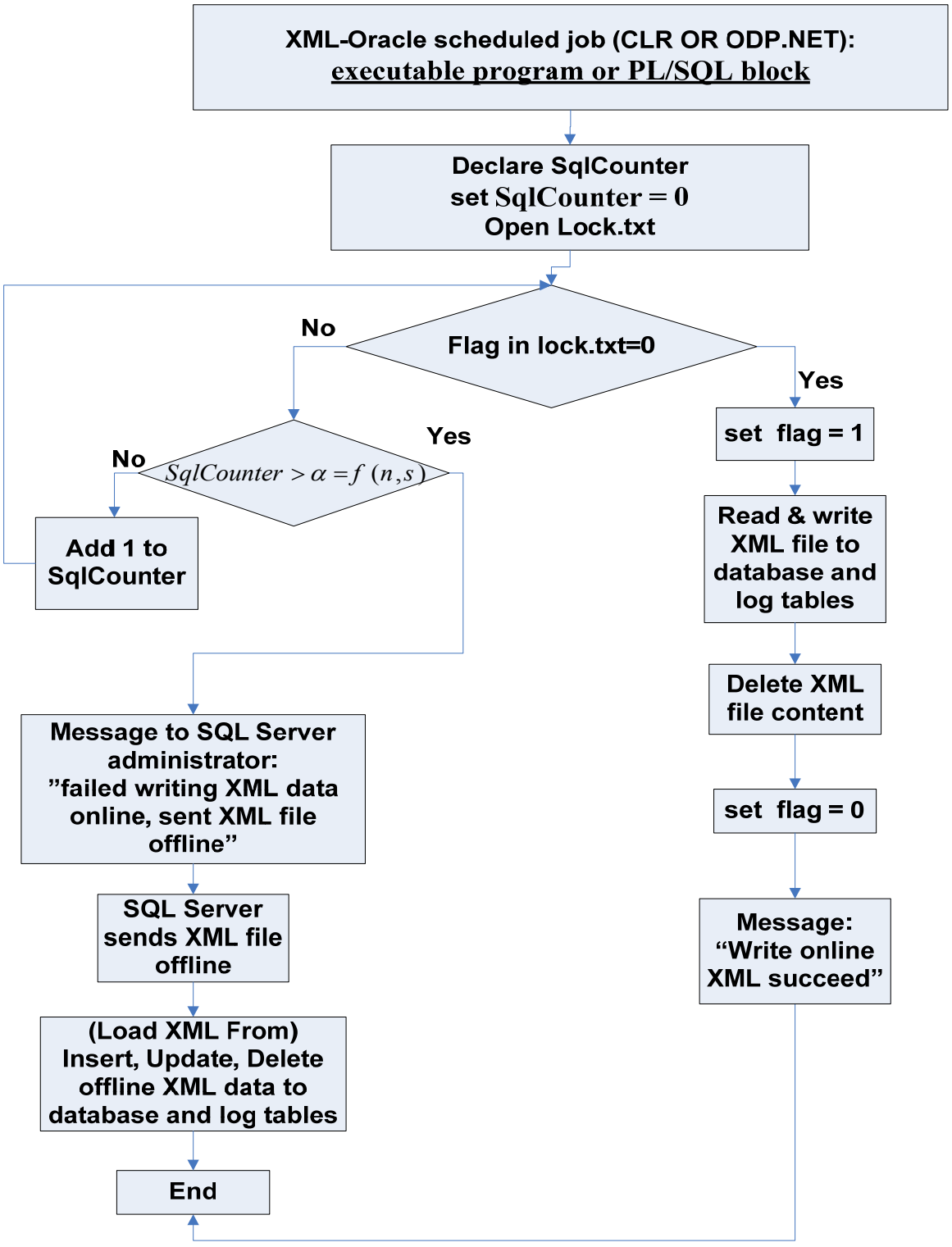


Figure 8: XML - Oracle Job Flow

www.ijric.org E-ISSN: 2076-3336

6. REFERENCES

[1] S. P. Paul, "*Pro SQL Server 2005 Replication"*, Apress, 2006, ISBN: 13- 978-1-59059-650-0

[2] S. Holzner, *"Inside XML"*, New Riders Publishing, 2001, First Edition, ISBN: 0-7357-1020-1

[3] J. Sebastian, *"The Art of XSD, SQL Server XML Schema Collections",* First published by Simple Talk Publishing, 2009, ISBN: 978-1-906434-13-7

[4] G. Malcolm, *"Programming SQL SERVER 2000 With XML"*, MsPress, 2002 Second Edition, ISBN: 0-7356-1774-0

[5] S. Klein, *"Professionsal SQL Server XML",* Wiley Publishing, 2006, ISBN-13: 978-0-7645-9792-3

[6] D. Adams, *"Oracle XML DB Developer's Guide, 10g Release 2 (10.2)",* Oracle 2005, B14259-02

[7] J. Melnick, *"Oracle XML Developer's Kit Programmer's Guide, 10g Release 3 (10.1.3)",* Oracle, 2006, B28236-01

[8] R. Dewson, and J. Skinner, *"Pro SQL Server 2005 Assemblies"*, Apress, 2006, ISBN: 1-59059-566-1

[9] J. C. Pulakhandam and S. Paruchuri, *"ODP.NET Developer's Guide, Oracle Database 10g Development with Visual Studio 2005 and the Oracle Data Provider for .NET"*, Packt Publishing, 2007, ISBN: 978-1-847191-96-0

BIOGRAPHY:

Dr. Seifedine Kadry received his masters' degree in Modeling and Intensify Calculus (2001) from the Lebanese University – EPFL - INRIA. He received his PhD (2003-2007) in applied mathematics from Blaise Pascal University, and IFMA, France. He worked as Head of Software Support and Analysis Unit of First National Bank. He published many papers in national and international journals. His research interests are in the areas of Computer Science, Numerical Analysis and Stochastic Mechanics. Since 2007, Dr Kadry is an associate professor at the Lebanese University.

Dr. Hussam Kassem received the masters' degree in Electrical & Electronics Engineering from the University of Science and Technology of Lille (USTL), France, in 1998. He received his Ph.D. in Telecommunications from CNAM (Conservatoire National des Arts et Metiers) of Paris, France in 2002. From 2002 till 2008, he was the coordinator of the CCE Department at the American University of Science and Technology (AUST) in Lebanon. Now, He teaches at the Lebanese University, Faculty of Science and at the Lebanese International University, Faculty of Engineering. He published many papers in national and international journals. His research interests are Antennas Processing, Channel Equalization, Multichannel Estimation, Signal Processing, and Mobile Communications.

Dr. Mohamed Smaili received the masters' degree in Computer Science from the University of Monpelier II (USTL), France, in 1990. He received his Ph.D. in Computer Science from the University of Monpelier II (USTL), France, in 1993. Since 1994 he teaches at the Lebanese University, Faculty of Science. He published many papers in national and international conferences. His research interests are fuzzy logic, digital circuits and simulation.