



InFraMegh -An Implementation of Integrated Framework for Cloud Based Transactional Applications in Educational Institutes

Indu Arora¹, Dr. Anu Gupta²

¹Research Scholar, Department of Computer Science and Applications,
Panjab University, Chandigarh, India

²Associate Professor, Department of Computer Science and Applications,
Panjab University, Chandigarh, India

Abstract: Relational Data Base Management Systems (RDBMS) has been in use for handling the business processes successfully almost for forty years. Now Companies are choosing Cloud Computing (CC) for managing and improving their business processes. NoSQL databases are very popular in Cloud for applications like Business Analytics that are mainly read-intensive analytical applications. Deployments of write-intensive transactional applications, managed traditionally by RDBMS, are not considered safe in Cloud due to stringent consistency requirements of data. So a need is felt to model and design an Integrated Framework for Data Management of Transactional Applications in Cloud named as InFraMegh to facilitate the development of transactional applications for educational institutes. InFraMegh uses In-Memory Data Grid as a middleware between application software and database to keep data consistent in Cloud. This paper illustrates mapping among architecture, model, detailed design and implementation of InFraMegh. It further gives details of its implementation by presenting code snippets of Java classes used in the implementation of InFraMegh. InFraMegh is implemented to simplify the integration efforts involved in the integration of different tools and development efforts of developers of transactional applications meant for Cloud.

Keywords: Cloud Computing, In-Memory Data Grid InFraMegh, Transactional Application

I. Introduction

Transactional applications are used almost in every sector for the purpose of fast, efficient and reliable data processing. Payroll and Personnel Information, Financial Accounting and Store-Keeping applications are the commonly used transactional applications. Educational sector is also making use of either in-house or third party transactional application software to manage its data. All institutes scattered across the wider area manage data differently and individually. There is no uniform way to represent, store, access and integrate data required by transactional applications. Due to lack of integration of data of various educational institutes, it is not possible to use their digital data for analytical processing. Deploying transactional applications in Cloud will help educational institutes in managing their data uniformly while optimizing cost and usage of their IT infrastructure.

ACID (Atomicity, Consistency, Isolation, Durability) properties ensure the reliable processing of database transactions. In-Memory Data Grid (IMDG) can be used as middle layer between application server and external database to ensure ACID properties of transactional applications in Cloud. Due to these observations and limitations, a need is felt to have a uniform and cost-effective solution to handle transactional data of educational institutes. The Integrated Framework for Data Management of Transactional Application in Cloud is proposed and named as InFraMegh. The term InFraMegh has been derived from the initial letters of **I**ntegrated **F**ramework and the **M**egh. Megh is a Hindi word used for Cloud [1].

The main significance of developing InFraMegh is to provide an integrated framework that can assist transaction application developer in developing and deploying transactional applications in Cloud. Cloud environment is different from traditional database scenario. Developer of Cloud based transactional applications has to ensure speed, high availability and ACID guarantees so that Cloud user can access such data and applications from any location through clients like thin computers, mobiles etc. [2],[3]. InFraMegh is developed using various tools each having different configuration settings. InFraMegh allows developer of transactional applications to make configuration settings through its Graphical User Interface. While implementing InFraMegh, IMDG concepts are used to bring data closer to the application that further enhances performance [4]-[6]. InFraMegh provides generalized APIs such as storing data in cache, inserting/deleting data from cache/database etc. to assist the developer of transactional applications to be deployed in Cloud.

This paper has been structured into seven sections. Second section briefs the design of InFraMegh. Third section presents implementation architecture and its mapping with architecture, model and design. Fourth section explains the tools used for implementation. Fifth, Sixth and seventh section discusses implementation of Configuration Management, Data Access Management and Database Layer respectively and includes code snippets of important activities. Eighth section discusses the File structure of InFraMegh followed by conclusion.

II. Background of Present Work

Four-layered architecture for deploying transactional applications in Cloud includes Cloud Infrastructure layer, Cloud Data Management layer, Virtualization layer and Cloud Interface layer [7]. Cloud Infrastructure layer includes distributed hardware components (such as server, storage and network devices) and host operating system having capability of managing these components. Cloud Data Management layer manages data of transactional applications in Cloud environment. Virtualization Layer prepares virtual machines comprising guest Operating System (OS), data stored in In-memory Data Grid (IMDG) and applications for the use of Cloud users.

Model and design of InFraMegh focuses on Cloud Data Management layer. High level model of InFraMegh comprises Configuration Management and Data Access Management components. The purpose of Configuration Management is to provide a facility for developers so that they can configure the ecosystem required for designing and developing transactional applications meant for Cloud. The ecosystem includes setting of database MySQL, middleware CloudTran and IMDG Oracle Coherence. The purpose of Data Access Management component is to provide general APIs for transactional applications to manage data stored in IMDG or database. Detailed Design of InFraMegh includes Database Design, Process Design and User Interface (UI) Design.

A. Database Design

This part of the InFraMegh has been designed keeping in view the settings of configuration or environment attributes. These attributes are required during implementation of InFraMegh and deployment of transactional applications in Cloud. Based on InFraMegh model, entities like *startCDMF*, *Configuration files*, *CDMSetting*, *Users* and *Rights* have been designed. *StartCDMF* entity store initial information like server name, IP address etc. *Configuration files* are XML files required by CloudTran. These files store information about all tables of related database. The purpose of *CDMSetting* is to update configuration files based on tables stored in related database and settings provided by developer through GUI of InFraMegh. *User* entity contains information about Cloud user while *Right* entity stores the privileges assigned to Cloud users.

B. Process Design

Process Design includes procedures for managing activities of Configuration Management and Data Access Management components of InFraMegh. Configuration Management deals with setting configuration environment. The processes involved in Configuration Management are further grouped into two categories. The processes required to get configuration related data from the developer of transactional applications are defined in *startCDMF* entity and processes required to perform tasks related to setting and updating of configuration attributes in XML files or database tables are defined in *generateClassSchema* entity. Data Access Management include processes required for managing data in IMDG and database and are part of *dbCacheInterface* entity.

C. User Interface Design

Graphical User Interface of InFraMegh is designed for getting configuration related values from the developer of transactional applications. These values are stored in configuration files. Fig. 1 illustrates inception of InFraMegh starting from generalized architecture of Cloud for transactional Applications to high level model and detailed design.

III. Implementation Architecture

Design of InFraMegh is implemented using a 3-layered architecture illustrated in Fig. 2. This architecture encompasses Presentation Layer, Cloud Data Management Layer and Database Layer. Presentation Layer corresponds to User Interface design. Cloud Data Management Layer deals with Data Management objects categorized further as Configuration Management Objects and Data Access Management Objects. Design of InFraMegh includes three entities: *startCDMF*, *generateClassSchema* and *dbCacheInterface*. These entities have been implemented in different layers of implementation architecture. Table 1 shows the mapping of design and implementation layers of InFraMegh.

Figure 1 InFraMegh- From Conception to Detailed Design

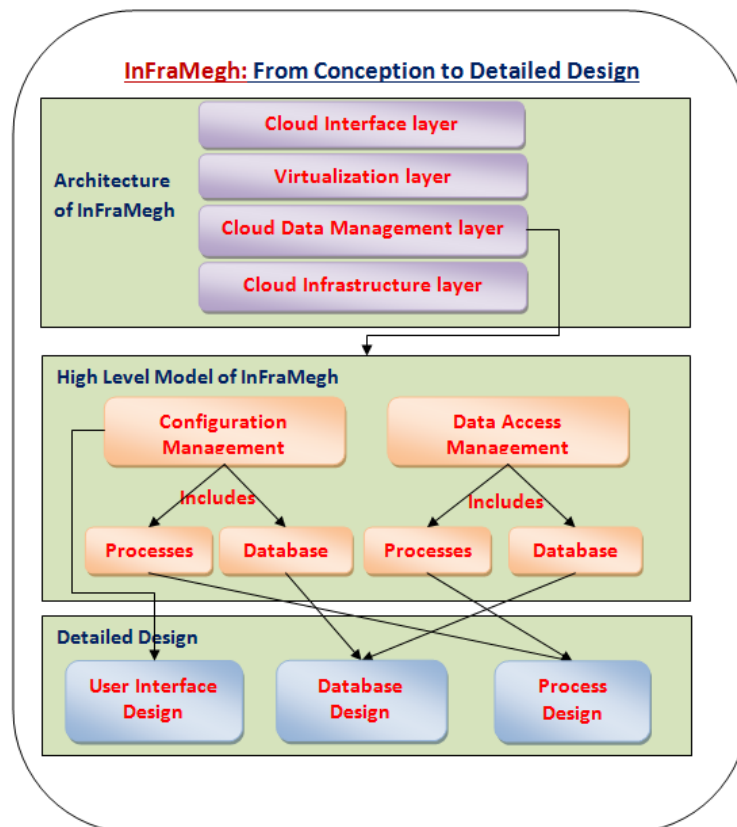


Figure 2 Three-layered Implementation Architecture of InFraMegh

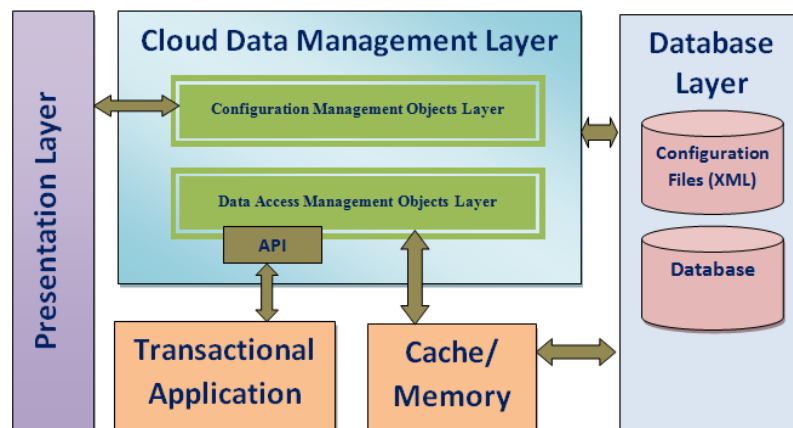


Table 1: Mapping of Implementation and Design of InFraMegh Components

Design Component →	Configuration Management		Data Access Management
Layer ↓	Start CDMF	Generate Class Schema	dbCache Interface
Presentation	✓		
Cloud Data Management		✓	✓
Database	✓	✓	✓

IV. Tools Used for Implementation

InFraMegh is built over CloudTran and Oracle Coherence. CloudTran, a middleware tool, is used to provide data consistency. It is built over Oracle Coherence and provides ACID guarantees for transactions [8]. Oracle Coherence is industry standard In-Memory Data Grid (IMDG) solution which provides backend support to CloudTran to manage data in grid having multiple nodes in cluster [9]. MySQL is used as backend database for InFraMegh. The database schema for InFraMegh is implemented using MySQL [10]. Java Enterprise Edition 1.7 is used for implementing the procedures of InFraMegh proposed in its detailed design. Java's main feature, Java Persistence API (JPA), is used to provide object/relational mapping facility for managing relational data in Java Applications [11]. Developers of transactional applications are required to design reports using Jasper iReport. InFraMegh includes an API that provides data to these pre-designed reports. These reports can be displayed in Jasper Viewer [12], [13].

V. Implementing Configuration Management

The *startCDMF* and *generateClassSchema* classes of Configuration Management are implemented in Presentation Layer and Cloud Data Management Layer respectively. The database design has been implemented in Database Layer. This implementation of database caters to both Configuration Management and Data Access Management components.

A. Implementing startCDMF in Presentation Layer

The purpose of *startCDMF* class is to create Graphical User Interface for developer of any transactional application. It has been implemented in Java using Swing controls and Delegation Event Model. This class allows developer to choose system settings such as default folder name, database and port number. It has 30 private/public methods and more than 150 variables/objects/instance variable declarations.

- **Part 'A'** is used to inherit *propertyChangeListener* in *startCDMF* class. It helps in displaying intermediate results in Progress bar during execution e.g. when developer initiates the process of initial configuration, it does various tasks like setting up of initial parameters in *startCDMF.xml*, configuring various configuration files, creating Java classes based on database tables and compiling Java classes. The partial implementation is shown below.

```
public class startCDMF implements PropertyChangeListener
{
    static class Task extends SwingWorker<Void, Void>
    {
        private ImageIcon icon1 = new ImageIcon("iconcdmf.jpg");
        private JFrame frame1 = new JFrame();
        private JProgressBar progressBar = new JProgressBar(0,2000);
        public Task() { dialog = new JDialog(frame, "Configuring InFraMegh Items", true); ... } ... }
}
```

- **Part 'B'** of the class file is called in *createAndShowGUI()* method to display GUI controls in seven different tabs named as Initial Settings/ Configuration, Insert Table, Remove Table, User Management, User Rights, Table Type and Backup/Restore. The partial implementation is shown below.

```
public static void addComponentsToPane(Container pane)
{
    JTabbedPane myPane = new JTabbedPane(JTabbedPane.TOP);
    ChangeListener changeListener = new ChangeListener()
    {
        public void stateChanged(ChangeEvent changeEvent)
        { JTabbedPane sourceTabbedPane = (JTabbedPane)
          changeEvent.getSource();
          createTabbedPane1(jTabbedPaneMyPane,pane);
          createTabbedPane2(jTabbedPaneMyPane,pane);
          createTabbedPane3(jTabbedPaneMyPane,pane);
          createTabbedPane4(jTabbedPaneMyPane,pane);
          createTabbedPane5(jTabbedPaneMyPane,pane);
          createTabbedPane6(jTabbedPaneMyPane,pane);
          createTabbedPane7(jTabbedPaneMyPane,pane);..... } .. }
}
```

- **Part 'C'** contains implementation of creating and showing various controls in one tab. The partial implementation is shown below.

```
private static void createTabbedPane1(JTabbedPane jTabbedPaneMyPane, Container pane)
{
    //declares controls
    Insets insets = pane.getInsets();
    lblhead1 = new JLabel("INITIAL SETTINGS/ CONFIGURATION"); ..... }
}
```

- **Part 'D'** helps in implementing actions depending on the event triggered by developer at any point of time. All seven tabs have different tasks and hence different event handlers are implemented to perform them. The partial implementation is shown below.

```
public void actionPerformed(ActionEvent e)
{
    if ("go".equals(e.getActionCommand()))
    { ..... }
}
```

- **Part 'E'** shows the main function of Java which is the starting point of execution whenever developer starts

developing transactional application using InFraMegh. The partial implementation is shown below.

```
public static void main(String[] args)
{
    javax.swing.SwingUtilities.invokeLater(new Runnable()
    {
        public void run()
        {
            createAndShowGUI();
            setValues();
        }
    });.....
}
```

B. Implementing generateClassSchema in Cloud Data Management Layer

In Cloud Data Management layer, *generateClassSchema* entity is implemented for all the actions or tasks required by InFraMegh. There is no Graphical User Interface (GUI) for this class. In presentation layer, *startCDMF* class creates object of this class and calls appropriate methods on occurrence of various events. A package is created to implement *generateClassSchema* entity. It consists of 47 private/public methods and 12 instance variables apart from more than 200 private variables/objects. The overall structure of the class is categorized into eight parts. Details of different parts of this class are listed below.

- **Part ‘A’** declares user defined package and imports about 27 system defined packages of Java. The partial implementation is shown below.

```
package cloudtran.cdmframe;
import java.sql.*;
import javax.swing.*;
```

- **Part ‘B’** gives instance variables and constructor for initialization of instance variables. The partial implementation is shown below.

```
public class generateClassSchema
{String filename=""; String ipadd="";Connection connect = null;
Statement statement = null;
public void setPara(String filename, String ipadd, String portno, String dbname, String username, String pwd,
String tname,String pathname)
{this.filename = filename; ....
}
```

- **Part ‘C’** creates Java classes based on database tables and compiles them automatically. The methods for this purpose are categorized into header, footer and column methods. Header methods write initial part of Java program like class declaration, import statements etc. Column methods declare each column of database table as instance variable in Java class file. They also write two methods, set and get for setting values and getting already stored values respectively. Footer methods include JPA methods such as *readExternal()* and *writeExternal()* for retrieving values from cache or saving values in cache. The partial implementation is shown below.

```
private void setHeader(BufferedWriter bw, String tname) throws IOException
{
    String content;
    content ="package cloudtran.cdmmmodel;";
    bw.write(content);
    bw.newLine();
    .... }
private void insert_col_details(BufferedWriter bw, String col_name, String col_type, int col_len, int
col_pos,ArrayList<String> readList, ArrayList<String> writeList, ArrayList<String> som) throws IOException
{String content;
...}
private void insert_col_methods(BufferedWriter bw, String col_name, String c_type, int col_pos ) throws
IOException
{String content;
content=" public static final int " + " POF_INDEX_" + col_name.toUpperCase() + " = " + col_pos + ";"; ....}
private boolean compileFile()
{try {String jcString = "javac -classpath .;" + pathname.substring(0,2)+ "//cdmf/classes;" + ".;./lib/*;"
classes//cloudtran// cdmmmodel/" + filename+".java"; ... }
.....
}
```

- **Part ‘D’** includes methods that fetch default values for configuration parameters from startCDMF.xml file and updates them. It makes use of nodes and nodelist objects using Java, for capturing values from other configuration files of XML type and saving them back appropriately. The partial implementation is shown below.

```

public String setInitialValues(String iname, String ival)
{ ...
DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
Document doc = docBuilder.parse(filepath);
Node perNode = doc.getFirstChild();
NodeList list_per = doc.getElementsByTagName("property");
for (int i = 0; i<list_per.getLength();i++) ...}
public String getInitialValues(String ival) { ...      Node inner_node =
doc.getElementsByTagName("property").item(i);}
private void getDBParameters()
{ try {
String cdmpath = getInitialValues("cdmpath");
String filepath = "startCDM.xml";
DocumentBuilderFactory docFactory2 = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder2 = docFactory2.newDocumentBuilder();
Document doc2 = docBuilder2.parse(filepath);
Node perNode2 = doc2.getFirstChild();
NodeList list_per2 = doc2.getElementsByTagName("property");
System.out.println("length is " + list_per2.getLength());
String actPath = getInitialValues("cdmpath");
ipadd = getInitialValues("serverIP"); ... }
}
}

```

- **Part ‘E’** includes implementation of updation of persistence.xml, a configuration file. One of the major functions of InFraMegh is to get values from developer through presentation layer and update configurations files provided by CloudTran, Coherence and InFraMegh. Other configuration files required by CloudTran are implemented in similar way. The partial implementation is shown below.

```

public void createPersistence() { try {
getDBParameters();....
sql = "select distinct c.table_name from information_schema.columns c where c.table_schema = '" + dbname + "'";
ResultSet resultset = statement.executeQuery(sql); ....
String sampleFile = cdmpath + "\\classes\\META-INF\\persistenceSample.xml";
String orgFile = cdmpath + "\\classes\\META-INF\\persistence.xml";
String jcString = "cmd /c copy " + sampleFile + " " + orgFile ;
Process p=Runtime.getRuntime().exec(jcString);
p.waitFor();
BufferedReader reader=new BufferedReader( new InputStreamReader(p.getInputStream()) );
...
DocumentBuilderFactory docFactory1 = DocumentBuilderFactory.newInstance();
DocumentBuilder docBuilder1 = docFactory1.newDocumentBuilder();
Document doc1 = docBuilder1.parse(filepath);
Node perNode1 = doc1.getFirstChild();
NodeList list_per1 = doc1.getElementsByTagName("property");
String actPath = getInitialValues("cdmpath");
for (int i = 0; i<list_per1.getLength();i++) {
Node inner_node = doc1.getElementsByTagName("property").item(i);
NodeList list2 = inner_node.getChildNodes();
NamedNodeMap attr = inner_node.getAttributes();
Node nodeAttr = attr.getNamedItem("name"); ... }
public void addTablePersistence(String tn)
{ try {
String cdmpath = getInitialValues("cdmpath"); ....
pu1.insertBefore(cl,pu1Prop); pu2.insertBefore(clRead,pu2Prop);....}
public void removeTablePersistence(String tn)
{ try {
String cdmpath = getInitialValues("cdmpath");
...
if (remNode1!= null)pu1.removeChild(remNode1);
if (remNode2!= null)pu2.removeChild(remNode2);
...
TransformerFactory transformerFactory2 = TransformerFactory.newInstance();
Transformer transformer2 = transformerFactory2.newTransformer();....
StreamResult result2 = new StreamResult(new File(filepath));
transformer2.transform(source2, result2);
}
}
}
}

```

- **Part ‘F’** includes implementation to provide metadata of database like table name, column name, etc. to calling methods.

```
public ArrayList columnListing(String db, String tb) {
    ArrayList<String> al = new ArrayList<String>();    try    {
        getDBParameters(); ...
        sql = "select distinct c.column_name from information_schema.columns c where c.column_name = 'id' and
        c.table_schema = '" + db + "'";
        sql += " and c.table_name = '" + tb + "'";
        ResultSet resultset = statement.executeQuery(sql); .....
        resultset.getString("column_name").toLowerCase();
        al.add(t_name);    }    ... }
```

- **Part ‘G’** includes implementation of tasks such as saving data related to user profile, rights assigned to user, marking master table etc. into database. The partial implementation is shown below.

```
public String userCreateSave(String userRightsFlag,long selectedId,long selectedVersion,String userName,String
userFull,String userAdd,String userCity,String userPhone,String userEmail,String userCP,String userCPD,String
userStatus,String userType) ....
public String userRightsSave(String userRightsFlag,long selectedId,long selectedVersion,String userName,String
userDB,String userTB,String userCol,    int userVal)
{...}
public String save_table_type(JTable mytable){ ...}
```

- **Part ‘H’** shows implementation for taking backup of entire database. The name of backup file is defined using current date. Restoring the backup file needs filename as parameter and uses database command to restore data. The partial implementation is shown below.

```
public String createBackup(String column_value)
{
    Date date = new Date();
    DateFormat dateFormat = new SimpleDateFormat("yyyyMMdd");
    String retString="";
    generateClassSchema gc = new generateClassSchema();
    String[][] table_list = gc.tableTypeListing();
    try    {
        String fn =pathname.substring(0,2)+"\\"+pathname.substring(2)+"\\"+column_value+"backup"
        +dateFormat.format(date)+".dmp";
        for(int cnt = 0 ; cnt < table_list.length ; cnt++)
        {
            String backupCommand="cmd /c mysqldump -u"+dbUser + " -p"+ dbPWD + " " + dbName;
            if ("true".equals(table_list[cnt][1]) == false) {
                backupCommand += " " + table_list[cnt][0] + " -w" + column_Name + "=" + column_value;
            }
            else
            {backupCommand += " " + table_list[cnt][0];    }
            if (cnt ==0)                backupCommand += ">" + fn;
            else                backupCommand += ">>" + fn;
            String jcString =backupCommand ;
            System.out.println(backupCommand);
            Process p=Runtime.getRuntime().exec(jcString);
            p.waitFor();
            BufferedReader reader=new BufferedReader(
            new InputStreamReader(p.getInputStream())    );    }
            retString="Backup file created in folder " + fn;
        } ...}
    public String restoreFull(String fileName)
    {String retString;
        try    {
            String restoreCommand ="cmd /c " +
            pathname.substring(0,2)+"\\"+pathname.substring(2)+"\\mysql\\bin\\mysql ";
            restoreCommand = restoreCommand + " -u"+dbUser + " -p"+ dbPWD + " " + dbName;
            restoreCommand += "<" + fileName;
            String jcString =restoreCommand ;
            System.out.println(restoreCommand);
            Process p=Runtime.getRuntime().exec(jcString);
            p.waitFor();
            BufferedReader reader=new BufferedReader(
            new InputStreamReader(p.getInputStream())    );
            retString="Dump file restored:- " + fileName;
        }    catch(Exception ex)    {    ex.printStackTrace();
            retString ="Error in restoring dump file";
        }
        return retString;    }
```

VI. Implementing Data Access Management

The methods designed for Data Access Management component are implemented in *dbCacheInterface* class used in Cloud Data Management Layer. The database required for this class is described further in Database Layer.

A. Implementing dbCacheInterface in Cloud Data Management Layer

The implementation purpose of *dbCacheInterface* class in Cloud Data Management Layer is to provide a facility to developer with generalized methods that can manage the data stored in In-memory Data Grid while maintaining ACID guarantees. It also provides methods for generating reports. It acts as a mediator between CloudTran and developer of transactional applications. It includes 7 instance variables and 13 private/public methods. The methods are divided into four parts. The Purpose of each part is given below.

- **Part ‘A’** shows the declaration of user defined package, import of system defined packages, initialization of instance variables and constructor. The partial implementation is shown below.

```
package cloudtran.cdmframe;
import com.tangosol.net.CacheFactory;
import com.tangosol.net.NamedCache;...
public class dbCacheInterface{
String puDirect=""; String dbName=""; String dbUser="";...
public dbCacheInterface(){
    generateClassSchema gc = new generateClassSchema();
    puDirect = gc.getInitialValues("pu-client");
    dbUser = gc.getInitialValues("javax.persistence.jdbc.user");
    ("javax.persistence.jdbc.password"); ....}
public void startCloudTranClient(){StartClient.init();.....}
```

- **Part ‘B’** defines methods that get data directly from database bypassing CloudTran. The partial implementation is shown below.

```
public Collection getTableData(String tname,String cri,Object o) {
    System.setProperty( "tangosol.pof.enabled", "true" );
    System.setProperty( "puNames", puDirect);
    System.setProperty( "tangosol.coherence.distributed.localstorage", "false" );
    EntityManagerFactory emf = Persistence.createEntityManagerFactory(puDirect);
    EntityManager entityManagerDB = emf.createEntityManager();
    NamedCache cache = CacheFactory.getCache(tname);try {
    Query query1 =entityManagerDB.createQuery("select cate from " + tname + " cate " + cri);Collection c =
    (Collection)query1.getResultList();
    return c; } ...public Collection getTableData(String tname,String cri) { ...
public Collection getTableDataFromCache(String tname,String cri,Object o){..... }
```

- **Part ‘C’** defines that part of package which contains implementation of data updation in cache such as saving data, retrieving data from cache and putting data from database to cache. It also provides methods for creating backup file. The partial implementation is shown below.

```
public void put_in_cache(String tname,Object o, Long v) {
    NamedCache cache = CacheFactory.getCache(tname);
    EntityManagerFactory emf1 = Persistence.createEntityManagerFactory( puCloudTran);
    EntityManager entityManager1 = emf1.createEntityManager();
    EntityTransaction transaction1 = null; try {
    transaction1 = entityManager1.getTransaction();
    entityManager1.persist(o);
    cache.put(v, o); // to put key,value in cache }
    public String remove_object(String tname,Long v, Class type){
    NamedCache cache = CacheFactory.getCache(tname);....}
    public String saveData(Object obj, String clsName, String tname){ System.setProperty( "tangosol.pof.enabled",
    "true" );
    System.setProperty( "puNames", puCloudTran );
    System.setProperty( "tangosol.coherence.distributed.localstorage", "false" );
    transaction.begin(); entityManagerDB.persist(obj); transaction.commit(); ...}
    public String createBackup(String column_value)
    {String backupCommand="cmd /c mysqldump -u"+dbUser + " -p"+ dbPWD + " " + dbName;... }
    .....
```

- **Part ‘D’** defines method for generating reports. Developer will design reports using Jasper Reports but data will be provided to these reports through API provided by InFraMegh. The partial implementation is shown below.

```
public void generateReport(String jasperFileName, String[][] data, String[] columnNames) {
    JasperPrint jasperPrint = null;
    DefaultTableModel tableModel = new DefaultTableModel(data, columnNames); try {
    jasperPrint = JasperFillManager.fillReport(jasperFileName, new HashMap(),
    new JRTableModelDataSource(tableModel));
    JasperViewer jasperViewer = new JasperViewer(jasperPrint,false);
    jasperViewer.setVisible(true); }
```

VII. Implementing Database Layer

Presentation Layer and Cloud Data Management Layer both require database and configuration files for storing and retrieving data during development of transactional applications using InFraMegh. The third layer named as Database Layer deals with the storage of database and configuration files. Implementation of both is described below.

A. Implementing Database Design

For current framework, three tables are designed in MySQL. *User_master* table stores the profile of Cloud users who are authorized to use transactional applications deployed in Cloud environment. Another table, *user_rights*, is created for storing information about rights assigned to users. The purpose of this table is to restrict the logical view of data according to the rights assigned to the Cloud user. The third table, *table_type*, stores type of table that can be either master or transaction. Based on these types, selective backup is taken for a Cloud user. The Data Definition Language (DDL) used for creating tables is given in Fig. 3.

Figure 3 Implementation of Database Design

<pre>Drop database if exists cdmdb;Create database cdmdb;Use cdmdb; Create table user_master (id bigint primary key, version long, User_name Varchar(15) not null, Full_name Varchar(100) Not null, Address Varchar(100), City Varchar(50), Phone int(10), Email Varchar (100), Contact_person Varchar(100), Contact_design Varchar(100), User_type varchar(20), user_status varchar(15));</pre>	<pre>Create table user_rights (id bigint primary key, version long, User_name Varchar(15), Db_name Varchar(15), Db_Table_name varchar(15), db_table_col_name varchar(15), col_value varchar(50)); create table tabletypes (tablename varchar(100), tabletype varchar(6));</pre>
--	---

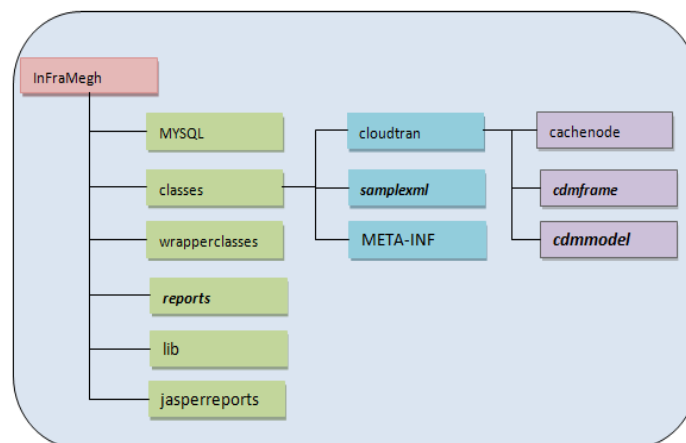
B. Implementing Configuration File Design

As per requirement of CloudTran, Oracle Coherence and InFraMegh, the configuration files are designed. Five configuration files are provided by CloudTran. One configuration file is designed and implemented for InFraMegh. The templates of configuration files provided by CloudTran are stored in a separate folder. During Initial Setting/Configuration, all the configuration files are copied to main folder and then updated based on related database table(s).

VIII. File Structure of InFraMegh

The file structure of InFraMegh is designed keeping in mind the logical similarities of different components. It contains mainly Java class files and data files. It also includes folders required by third party tools such as CloudTran, Oracle Coherence, MySQL and Jasper Reports. Folder *MYSQL* stores files related to MySQL database. Folder named as *classes* includes subfolders named as *cloudtran*, *META-INF* and *samplxml*. Folders *cloudtran* and *META-INF* are required by *cloudtran* while *samplxml* folder is used by InFraMegh for keeping templates of all configuration files. The actual configuration files are stored in *classes* folder. *META-INF* contains *persistence.xml* file. Subfolder *cloudtran* further includes subfolders *cachemodel*, *cdmframe* and *cdmmodel*. Subfolder *cachemodel* is required by middleware CloudTran for managing cache and cluster, while *cdmframe* and *cdmmodel* folders are used for keeping InFraMegh files. Folder *cdmframe* includes *generateClassSchema* and *dbCacheInterface* while *cdmmodel* contains Java classes based on database tables which are created automatically using InFraMegh. *StartCDMF* file lies in main folder named as *CDMF*. Folder *lib* contains all the jar files provided by CloudTran and MySQL. Folders like *wrapperclasses*, *lib* are required by CloudTran while *jasperreports* folder stores the library provided by Jasper Reports, a reporting tool. Folder named as *reports* contains reports created in Jasper Reports. Figure 4 shows the hierarchy of file structure of InFraMegh.

Figure 4 File Structure of InFraMegh



IX. Conclusion

The proposed design of InFraMegh is implemented using 3-layered architecture that encompasses Presentation Layer, Cloud Data Management Layer and Database Layer using tools like Java, CloudTran, Oracle Coherence and MySQL. MySQL provides ACID guarantees but lacks scalability. So, CloudTran and Oracle Coherence are used for better performance and to provide ACID guarantees to transactional data stored in the RAM of different servers in Cloud. To make them work in synchronization, a data management layer is developed which does all the necessary integration of different technologies and provides common APIs to manage data operations. Though different commercial products are there, but they work in isolation. So InFraMegh is implemented to reduce the complexities in developing transactional applications for Cloud and to provide help to developers in developing transactional applications for Cloud for any domain with minor modifications. The future work will include verification of implemented framework using an application of educational institutes as proof of concept.

References

- [1] Indu Arora and Anu Gupta "Modeling and Designing Integrated Framework for Data Management of Transactional Applications in Cloud," International Journal of Scientific & Engineering Research, vol. 6, Issue 1, ISSN 2229-5518, 2015, pp. 428-433.
- [2] Indu Arora and Anu Gupta, "Cloud Databases: A Paradigm shift in Databases," International Journal of Computer Science Issues (IJCSI), Vol. 9, Issue 4, No. 3, 2012, pp. 77-83.
- [3] Daniel J. Abadi, "Data Management in the Cloud: Limitations and Opportunities", Bulletin of the IEEE Computer Society Technical Committee on Data Engineering: vol. 32, No. 1, pp. 3-12, 2009
- [4] Indu Arora and Anu Gupta, "Usage of In-Memory Data Grids in Deployment of Transactional Applications in Cloud", In Proceedings of the 7th International Conference on Advanced Computing and Communication Technologies, 2013.
- [5] Indu Arora and Anu Gupta, "Improving Performance of Cloud based Transactional Applications using In-Memory Data Grid," International Journal of Computer Applications, ISSN No 0975- 8887, vol 107, No 13, 2014, pp. 14-19.
- [6] Barkha Bahl, Vandana Sharma and Navin Rajpal, "Boosting Geographic Information System's Performance using In-Memory Data Grid", BVICAM's International Journal of Information Technology, vol. 4, No. 2, 2012, pp. 468-471.
- [7] Indu Arora and Anu Gupta, "An Integrated Framework for Cloud Data Management in Educational Institutes," International Journal Computer Technology & Applications, ISSN No 2229-6093, Vol 5 (5), 2014, pp.1651-1657.
- [8] "CloudTran Technology Genesis," A Technology Whitepaper by CloudTran Inc, 2011, Available: CloudTranTechnologyBrief.pdf [Accessed: July10, 2015]
- [9] "Developing Applications with Oracle Coherence," Available: <https://docs.oracle.com/middleware/1212/coherence/COHDG.pdf> [Accessed : July 12, 2015]
- [10] "Java Persistence API (JPA) Tutorial," Available: http://www.tutorialspoint.com/jpa/jpa_tutorial.pdf [Accessed: August 23, 2015].
- [11] "MySQL 5.7 Reference Manual - MySQL Community," Available: [downloads.mysql.com /docs/refman-5.7-en.pdf](http://downloads.mysql.com/docs/refman-5.7-en.pdf) [Accessed: June 13, 2015].
- [12] "Jasper Reports - Quick Guide," Available: http://www.tutorialspoint.com/jasper_reports/pdf/jasper_quick_guide.pdf [Accessed: June 10, 2015].
- [13] "The Jasper Reports Ultimate Guide," <http://jasperreports.sourceforge.net/JasperReports-Ultimate-Guide-3.pdf> [Accessed: June 10, 2015].