Journal of Engineering and Applied Sciences 13 (10): 3358-3363, 2018
ISSN: 1816-949X

Enhancement of Business Continuity Through Effective Database Integration and Database Gateway Implementation

[1]Young-Ho Shin and [2]Jae-Cheol Ryou
[1]Department of Information System Operation,
Korea Institute of Science and Technology Information, 245 Daehak-ro,
34141 Yuseong-gu, Republic of Korea
[2]Department of Computer Science and Engineering, Chungnam National University,
99 Daehak-ro, 34134 Yuseong-gu, Republic of Korea

Abstract: This study seeks to develop optimal measures that enable organizations or institutions to quickly resume web services after performing database integration and migration following the new establishment or modification of systems. Database integration was performed for multiple databases and instances by avoiding schema collision for each character set and the database connection information of various Java-based web services was analyzed. Using the information of new databases integrated with existing databases, connection information mapping tables were derived for each user account. The mapping tables were used to design and implement a conversion gateway for connection to new databases and web service migration was carried out. When multiple web services require migration due to a change in the target database, the corresponding database connection information of application programs for each service must be analyzed and services can undergo migration after connection information is modified. This involves significant time and costs. This study performed service migration by implementing a driver-level conversion gateway corresponding to the connection process of databases without having to separately modify application programs for each service. This method of migration allows organizations to resume web services in a shorter period of time. This quick restoration of services contributes to service stability and business continuity, thereby greatly enhancing business competitiveness. By implementing driver-level common functions for database connection resulting from a change in the database server, fast and stable standard measures were presented for database connection during migration.

Key words: Database integration, web service, migration, business continuity, Java database connectivity, DBCP

INTRODUCTION

The Korea Institute of Science and Technology Information (KISTI) is a government-funded research institute launched to enhance national competitiveness through systematic establishment of national research and development infrastructure. As a leader of advance R&D, KISTI provides local businesses and research institutes with science and technology information collected from around the world (Anonymous, 2016).

Based on its vast collection of data, KISTI offers a science and technology information search service via a professional search engine and various online services delivering high-quality information. These information services can be categorized into services initiated by KISTI and services commissioned by government agencies or private enterprises. Regardless of the service type, KISTI seeks to pool its resources and establish a joint system for integrated management, thereby ensuring greater efficiency in its operations. To achieve the aforementioned goal with minimal investment, the institute has established a system that enables computing resources to be shared and jointly utilized by each service type.

The fundamental responsibility of the department in charge of operating the information system is to manage various hardware such as servers, storage system, network and backup as well as key system software such as the web, Web Application Servers (WAS) and databases. The development of application programs to provide web services, related operations and maintenance is performed by the department in charge of the corresponding web services.

Corresponding Author: Jae-Cheol Ryou, Department of Computer Science and Engineering, Chungnam National University, 99 Daehak-ro, 34134 Yuseong-gu, Republic of Korea

However, when establishing a new information system or performing integration, the department in charge of operating the information system assumes responsibility for hardware and software (servers, applications, etc.) and carries out system integration and service migration. In this case, the most difficult process is service migration which is necessary with database integration. This involves modification or redevelopment of database linkages to process the business logic of each service, so as to accommodate changes arising from database integration. Fortunately, about 95% of KISTI's web services have been developed using Java.

Java is a highly portable language as it is independent of systems and platforms. It has gained popularity as World Wide Web Software due to its simplicity, reliability and stability (Chaudhary, 2014; Kogent Kearning Solutions Inc., 2014; Ying Bai, 2011).

As mentioned earlier, this study presents methods for database integration and efficient service migration in an environment where various web services rely on sharing-based database instances. When migrating various services, it is important to ensure business continuity by completing the migration in the shortest possible time.

This study presents methods of migrating web services in the shortest possible time by modifying database linkages while maintaining the application program environment.

MATERIALS AND METHODS

General migration process and procedures: For most organizations operating information systems, large-scale system reorganization and establishment occur at regular intervals as system environment demands have been designed to meet rapidly changing requirements. From the perspective of information technology, migration generally refers to the process of an operating environment moving to a new, more optimal operating environment. Migration covers a wide range of subjects, including hardware, software, database, data, business logic and application programs (Kazzaz and Rychly, 2015; Hao *et al*., 2006; Kun *et al*., 2007).

The new establishment or reorganization of computing resources which are the primary infrastructure needed to provide web services is accompanied by web service migration. Service migration is performed according to the procedures given in the Table 1.

Characteristics and advantages/disadvantages by JDBC driver type: The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases (JDBC driver, JDBC and UCP). JDBC drivers are available for most database platforms, from a number of vendors and in a number of different flavors. JDBC driver can be largely categorized into four types. The driver types differ in their method of implementation, system independence and performance (Crawford *et al*., 2002; Yang *et al*., 1998). The characteristics of the four types are given below with a review of their advantages and disadvantages.

Type 1 driver (JDBC-ODBC bridge): All commands through the JDBC driver are converted to the ODBC-type and sent to the ODBC driver. In this case, the application used by JDBC and the ODBC driver connected to the JDBC via. the JDBC-ODBC bridge must exist in the same system (Anonymous, 2017).

Advantage: Since, the JDBC-ODBC driver has many ODBC drivers, most can be used in the database system. This type is especially useful if the ODBC driver is installed on the client side in advance.

Disadvantage: There may be some delay caused by converting a command received through JDBC into ODBC. This driver type is not suitable for applications requiring fast performance.

Type 2 driver (Native-API): All commands through the JDBC driver are converted and sent according to system calls of DBMS systems such as Oracle and Sybase. The DBMS linkage is written in a native code like C/C++ and the JDBC driver can be implemented by wrapping with Java.

Advantage: The database linkage is implemented using native code and thus offers a faster speed than the JDBC-ODBC bridge.

Table 1: Migration procedures

Evaluation and assessment	Migration	Verification and application
Requirements analysis and scope determination	Detailed design of migration measures and establishment of strategies	Migration inspection and test
Analysis of existing environment	Schema migration	Optimization
Generation of evaluation reports	Data migration	
Risk factors and technical analysis	Business logic migration	
	Application program migration	

Disadvantage: Each client intending to use the JDBC driver must have the database library of the DBMS vendor installed.

Type 3 driver (network-protocol (middleware driver): All commands sent by the JDBC driver are sent to the middleware where they are converted into suitable commands and sent to the database. The results are sent back to the JDBC driver. This is a typical three-tiered server-client model structure.

Advantage: Since, the database is approached through the middleware there is no need to install a native library of ODBC driver database on the client side. Database connection can be controlled through the middleware thus enabling optimization of performance, expandability and portability.

Disadvantage: A middleware server must be implemented and linkages must be available for each database vendor.

Type 4 driver (database-protocol): This is the most common JDBC driver, implemented 100% in Java. Requests are sent directly to each DBMS via. the network. Most DBMS vendors provide type 4 drivers at no charge.

Advantage: The driver is easy to distribute as it does not require any installation of DBMS native library or middleware.

Disadvantage: Each DBMS must use a different JDBC driver. Among the four types, type 2 and type 4 drivers are often provided by vendors. Type 2 JDBC drivers require SQL*NET for use in Oracle and are connected to the database through native modules in hardware-dependent formats such as dll and so. In Oracle, JDBC type 2 drivers are known as Oracle Call Interface (OCI) drivers.

On the other hand, type 4 JDBC drivers can be connected to the database using only Java packages. In Oracle, these drivers are called thin drivers. They provide faster performance than type 2 drivers which require native modules like SQL*NET (for Oracle). Type 4 drivers can connect to the database using Java classes alone and support database connection for any hardware and operating system. However, their performance is poor than that of OCI drivers.

In Oracle, the libjdbc*. So, file is needed to use OCI drivers and the corresponding directory must be registered in the shared library path. That is, the Oracle client product must be installed in the server.

Despite the constraints involved in using OCI drivers they are considered easier to maintain on the client side, since, connection URLs do not have to be changed in application programs of client machines and the only modification required is for the Tnsnames.ora file installed in client machines.

DBCP (Database Connection Pool): Creating a new connection for each user can be time consuming (often requiring multiple seconds of clock time), in order to perform a database transaction that might take milliseconds. Opening a connection per user can be unfeasible in a publicly-hosted internet application where the number of simultaneous users can be very large.

Accordingly, developers often wish to share a "pool" of open connections between all of the application's current users. The number of users actually performing a request at any given time is usually a very small percentage of the total number active users and the only time that a database connection is required is during request processing. The application itself logs into the DBMS and handles any user account issues internally (the DBCP component).

While DBCP can place a burden on the WAS server, it is able to provide rapid database connection to users upon request. When targeting services at multiple users this method is effective as the relevant connections can be reused.

Analysis of database connection type: The method of connecting to an Oracle database through Java can vary depending on the use of thin drivers or OCI drivers as shown in Table 2.

Driver formats as well as driver types can be described in various forms for each application. Out of more than 200 science and technology information services, about 95% of web services are using Oracle databases. The compositions of the two connection types are given in Table 3 based on an analysis of the existing environment.

Table 2: JDBC driver formats

Thin driver formats	OCI driver formats
Oracle JDBC thin using a service name: jdbc:oracle:thin@//<host>: <port>/<service_name> Example:jdbc:oracle:thin@// 203.xxx.xxx.xx1:1521:srvn1 Oracle JDBC Thin using an SID: jdbc:oracle:thin@<host>:<port>:<SID> Example:jdbc:oracle:thin@ 203.xxx.xxx.xx2:1511:inst2 Oracle JDBC Thin using a TNS_name: jdbc:oracle:thin@<TNS_name> Example:jdbc:oracle:thin:@tns1	Oracle JDBC OCI driver format jdbc:oracle:oci@ <database_name> especially oci8 Jdbc:oracle:oci8:@ <database_string>

Table 3: Connection type and composition

Variables	JDBC	DBCP
Type 2 (OCI)	34	50
Type 4 (Thin)	104	2
Total	138	52

ServiceName, a new function offered in Oracle 8i and later, allows databases to be registered with listeners. Once databases are registered, Service_Name can be used as a parameter in Tnsnames.ora. SID is the unique name of the database and ServiceName is the alias used when connecting to database. TNS (Transparent Network Substrate) is a computer networking technology unique to Oracle and supports homogeneous peer-to-peer connectivity on top of other networking technologies such as TCP/IP, SDP and named piped. TNS is usually operated for connection to Oracle databases (Transparent Network Stustrate). A TNS name is the name of the entry in tnsnames. ora file which is kept in $ORACLE_HOME/network/admin directory. This file contains the information that is used by the system to connect to the Oracle database. Using this, a client can fetch server associated information transparently. tnsnames.ora file contains information such as protocol, host IP address, Port number, SID and server.

Preconditions and related tasks: The point of establishing preconditions and related tasks is the point at which physical migration of the web and WAS server is complete and the linkage between the database server and WAS server is being modified. The required preconditions are as follows:

- Database integration
- No change in database user ID and password
- No modification of source code for applications of the WAS server
- Same character set as existing schema
- Integration into new instance is case of any duplicate schema

When using OCI drivers, the alias information corresponding to connection_URL in the Tnsnames. ora file found in the TNS_ADMIN directory on the client side is modified according to information in the mapping table. When using thin drivers, the IP address and port of the database server and SID information of the corresponding database must be described. As such, a function must be added to the gateway library to convert relevant mapping information to the IP address, port and SID (Table 4).

Design of database connection conversion gateway: In oracle, JDBC type 2 connection and type 4 connection are connected to the database server through Oracle JDBC drivers. Web services that rely on OCI driver connections can be easily migrated by modifying the service alias. However, those that involve thin drivers must have all connections of the source code individually modified.

Table 4: Schema mapping table

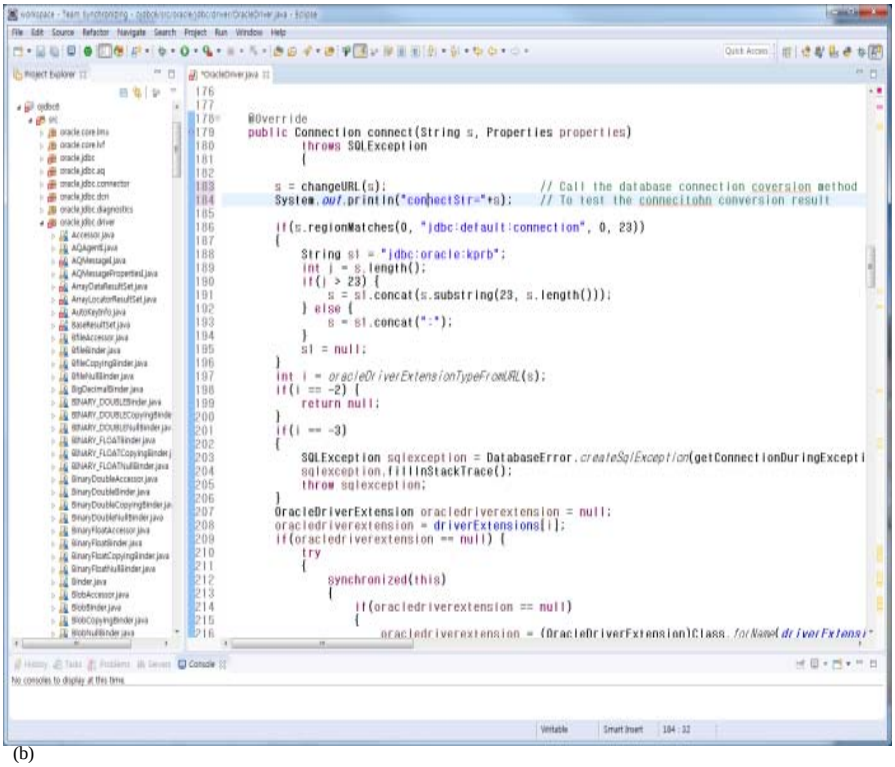
Schema	Old-instance	New-instance
User 1	O-INST1	N-INST1
User 2	O-INST2	N-INST2
...	O-INST2	N-INST3
User N	O-INST12	N-INST2

The modification of connection information from all web applications involves significant time, manpower and costs. This study proposes a database connection gateway that provides connection information of the new database before database connection. In the ojdbc driver, the Java.sql. Driver class is the implementation of the OracleDriver class. All connections using JDBC are generated through the public connection connect (String s, properties properties) method of the oracle.jdbc.driver. Oracle Driver class. This connection method was used to implement a conversion gateway method based on mapping information from the existing database target to the new database target. The gateway method returns a new connection_URL which supports connections to the new integrated database. This is the key to successfully establishing connections to the new integrated database.

Implementation, test and distribution: Based on the analytical results, a database conversion gateway was designed. This gateway was integrated into the JDBC driver package for optimized gateway application. The implementation procedures are as follows:

- Download the appropriate version of the JDBC library package for each service
- Unzip the downloaded JDBC library package
- Download two other decompile programs and decompile the package
- Double-check the decompile results
- Generate a project in the development platform (ie. Eclipse) and import the decompiled source
- As the Java compiler, use a Java Development Kit (JDK) compatible with the JDBC driver
- Summon the conversion gateway method in the connect method of the OracleDriver.java file (Fig. 1)
- Implement the database conversion gateway method
- Compile the modified package and replace the unzipped files in step 2 with the compiled OracleDriver.class and Oracle.jdbc.driver. OracleDriver.class
- Re-compress the modified JDBC library package
- Test the JDBC library package and check the connection to the modified database
- Distribute to the corresponding WAS server if no problems are found in test results

(a)

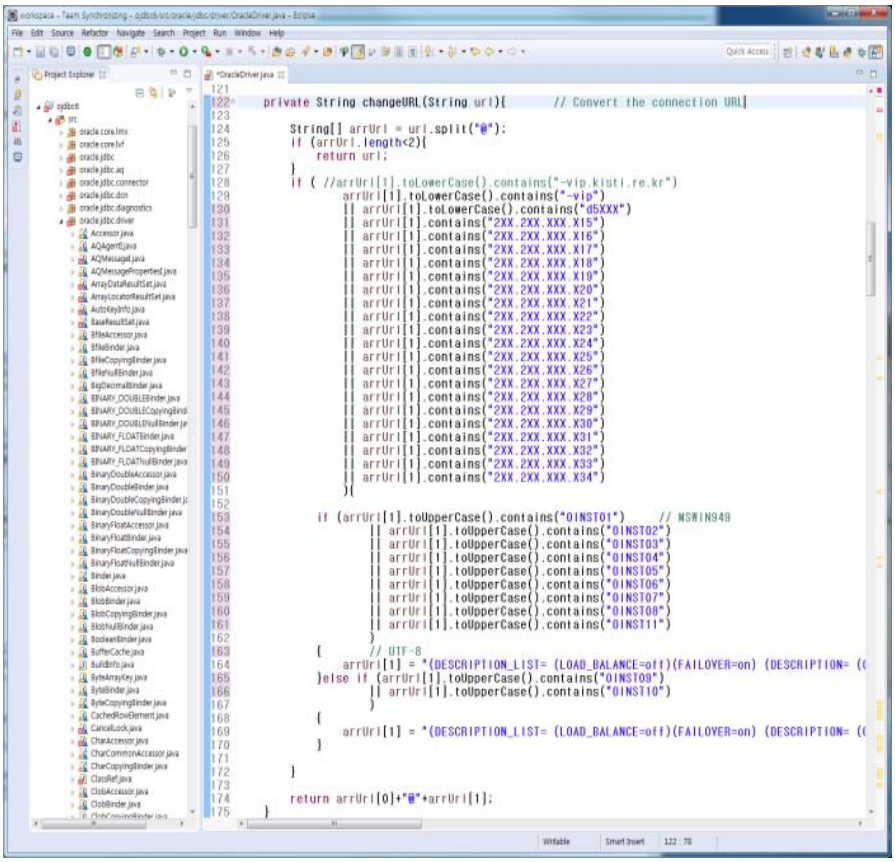


```
@Override
public Connection connect(String s, Properties properties)
        throws SQLException
        {

    s = changeURL(s);                          // Call the database connection coversion method
    System.out.println("connectStr="+s);       // To test the connectionn conversion result

    if(s.regionMatches(0, "jdbc:default:connection", 0, 23))
    {
        String s1 = "jdbc:oracle:kprb";
        int j = s.length();
        if(j > 23) {
            s = s1.concat(s.substring(23, s.length()));
        } else {
            s = s1.concat(":");
        }
        s1 = null;
    }
    int i = oracleDriverExtensionTypeFromURL(s);
    if(i == -2) {
        return null;
    }
    if(i == -3)
    {
        SQLException sqlexception = DatabaseError.createSqlException(getConnectionDuringExcepti
        sqlexception.fillInStackTrace();
        throw sqlexception;
    }
    OracleDriverExtension oracledriverextension = null;
    oracledriverextension = driverExtensions[i];
    if(oracledriverextension == null) {
        try
        {
            synchronized(this)
            {
                if(oracledriverextension == null)
                {
                    oracledriverextension = (OracleDriverExtension)Class.forName(driverExtensi"
```

(b)



```
private String changeURL(String url){       // Convert the connection URL

    String[] arrUrl = url.split("@");
    if (arrUrl.length<2){
        return url;
    }
    if ( //arrUrl[1].toLowerCase().contains("~vip.kisti.re.kr")
          arrUrl[1].toLowerCase().contains("~vip")
          || arrUrl[1].toLowerCase().contains("d5XXX")
          || arrUrl[1].contains("2XX.2XX.XXX.X15")
          || arrUrl[1].contains("2XX.2XX.XXX.X16")
          || arrUrl[1].contains("2XX.2XX.XXX.X17")
          || arrUrl[1].contains("2XX.2XX.XXX.X18")
          || arrUrl[1].contains("2XX.2XX.XXX.X19")
          || arrUrl[1].contains("2XX.2XX.XXX.X20")
          || arrUrl[1].contains("2XX.2XX.XXX.X21")
          || arrUrl[1].contains("2XX.2XX.XXX.X22")
          || arrUrl[1].contains("2XX.2XX.XXX.X23")
          || arrUrl[1].contains("2XX.2XX.XXX.X24")
          || arrUrl[1].contains("2XX.2XX.XXX.X25")
          || arrUrl[1].contains("2XX.2XX.XXX.X26")
          || arrUrl[1].contains("2XX.2XX.XXX.X27")
          || arrUrl[1].contains("2XX.2XX.XXX.X28")
          || arrUrl[1].contains("2XX.2XX.XXX.X29")
          || arrUrl[1].contains("2XX.2XX.XXX.X30")
          || arrUrl[1].contains("2XX.2XX.XXX.X31")
          || arrUrl[1].contains("2XX.2XX.XXX.X32")
          || arrUrl[1].contains("2XX.2XX.XXX.X33")
          || arrUrl[1].contains("2XX.2XX.XXX.X34")
          ){

        if (arrUrl[1].toUpperCase().contains("01NST01")     // MSWIN949
                || arrUrl[1].toUpperCase().contains("01NST02")
                || arrUrl[1].toUpperCase().contains("01NST03")
                || arrUrl[1].toUpperCase().contains("01NST04")
                || arrUrl[1].toUpperCase().contains("01NST05")
                || arrUrl[1].toUpperCase().contains("01NST06")
                || arrUrl[1].toUpperCase().contains("01NST07")
                || arrUrl[1].toUpperCase().contains("01NST08")
                || arrUrl[1].toUpperCase().contains("01NST11")
                )
        {       // UTF-8
            arrUrl[1] = "(DESCRIPTION_LIST= (LOAD_BALANCE=off)(FAILOVER=on) (DESCRIPTION= ((
        }else if (arrUrl[1].toUpperCase().contains("01NST09")
                || arrUrl[1].toUpperCase().contains("01NST10")
                )
        {
            arrUrl[1] = "(DESCRIPTION_LIST= (LOAD_BALANCE=off)(FAILOVER=on) (DESCRIPTION= ((
        }

    }
    return arrUrl[0]+"@"+arrUrl[1];
}
```

Fig. 1: a, b) Implementation of database connection URL conversion function

The above database conversion gateway replaces existing files in the Java library package path of the WAS server. For changes to TNS information used by OCI and DBCP, the tnsnames.ora file in the Oracle client installation directory or TNS_ADMIN path is modified with the updated database information.

RESULTS AND DISCUSSION

The assessment of database accounts, schemas and database connection types for more than 200 web services was time consuming and manpower intensive. The database connection types of web services exist in diverse forms with one service having one to four connections. Some connections comprised different drivers such as OCI, thin drivers and DBCP drivers. These diverse forms are seen as the result of adding new functions over time ombined with individual preferences.

This study performed service migration using a database conversion gateway implemented and tested based on analytical results. The key is to reflect the connection information to the new integration database server for web service components such as the web, WAS and database server.

To reflect the connection information to the new integration database server, a database conversion gateway was implemented at the driver level. In the WAS server, service migration becomes complete by modifying the information in the TNS_NAMES.ora file and replacing the corresponding JDBC driver.

The proposed method allowed migration to be completed in a few hours which is a significant improvement compared to the lengthy migration process for individual services.

CONCLUSION

KISTI collects a vast amount of science and technology information and provides extensive services. Organizations responsible for establishing large databases and offering various related services are burdened by database integration and migration which result from the new establishment or modification of systems.

When following the regular service migration process, it is almost impossible to resume web services within a few hours of database integration or migration. As such, the proposed method is expected to serve as an effective solution in circumstances where database integration or service migration must be performed in the shortest possible time.

While this study limited database integration and gateway implementation to Oracle databases they can be applied similarly to other databases operating under a Java-based JDBC environment such as MS-SQL and DB2. Factors to be considered are security risks and management issues related to the exposure of information in old and new databases for modification to connection URL in ojdbc drivers. Further research is needed to address security weaknesses. The results of this study were utilized in the quick migration of more than 200 web services being offered by KISTI. This helped to maintain the institute's business continuity and web service competitiveness.

REFERENCES

Anonymous, 2016. Transparent network substrate. Wikimedia Foundation Inc, San Francisco, California.

Anonymous, 2017. What's new in JDBC and UCP in Oracle database 12c release 2 (12.2)?. Oracle, Redwood City, California. http://www.oracle.com/technetwork/database/features/jdbc/index.html.

Bai, Y., 2011. Practical Database Programming with Java. Wiley/IEEE Press, Hoboken, New Jersey, USA., ISBN:978-0-470-88940-4, Pages: 952.

Chaudhary, H.H., 2014. Core Java Professional: Advanced Features (Core Series) Updated to Java 8. 2nd Edn., CreateSpace, New York, USA., ISBN:9781502370525, Pages: 582.

Crawford, W., J. Farley and D. Flanagn, 2002. Java Enterprise in a Nutshell: A Desktop Quick Reference. 2nd Edn., O'reilly Media, Sebastopol, California, Pages: 935.

Hao, W., T. Gao, I.L. Yen, Y. Chen and R. Paul, 2006. An infrastructure for web services migration for real-time applications. Proceedings of the 2nd IEEE International Workshop on Service-Oriented System Engineering (SOSE'06), October 25-26, 2006, IEEE, Shanghai, China, pp: 41-48.

Kazzaz, M.M. and M. Rychly, 2015. Web service migration using the analytic hierarchy process. Proceedings of the IEEE International Conference on Mobile Services (MS), June 27-July 2, 2015, IEEE, New York, USA., ISBN:978-1-4673-7284-8, pp: 423-430.

Kogent Learning Solutions Inc., 2014. Java Server Programming Java EE7 (J2EE 1.7), Black Book. Wiley, New Delhi, India, ISBN-13:9789351194170, Pages: 1304.

Kun, H., L. Chaohua, Z. Hua and H. Jun, 2007. Efficient web service migration algorithm. J. Comput. Appl., 24: 64-67.

Yang, A., J. Linn and D. Quadrato, 1998. Developing integrated web and database applications using JAVA applets and JDBC drivers. Proceedings of the ACM 29th Technical Symposium on Computer Science Education Vol. 30, February 26-March 01, 1998, ACM, Atlanta, Georgia, pp: 302-306.