Chapter 5

Integration of Legacy and New Applications for SMEs

5.1 Introduction

In view of the problems faced by Small and Medium Enterprises (SMEs) in developing countries regarding their Legacy Systems Integration, we propose a strategy for enterprise application integration for such a group. The strategy has following features:

1. It is based on all shareware components, like J2EE, JMS, and XML, all available freely on internet
2. Since majority SMEs work on LAN platforms, this platform is considered
3. SME applications like mini ERP, Open Office, Financial Accounting systems are considered. These all form different legacy systems.
4. The Affordable Solution Lab. At KReSIT, IIT Bombay (www.it.iitb.ac.in/asl/) is engaged in this activity.
5. It was tested with an experimental setup.

5.2 Issues in Application Integration for SMEs

Legacy applications have traditional platforms. These are typically developed independent of each other, by different set of programmers. The operating system platforms and front-end/ back-end softwares are chosen as per the suitability of the then team of programmers. This results in a typical set of application components/ modules as listed below:
Fig. 5.1 shows a typical SME layout. These legacy system modules are complete monolithic softwares and have their own user interface, data entry/query screens, transactions and reports. These modules are self-sufficient. In this complex and dynamic environment, homogeneous IS architectures are no longer practical options.
However integration across any two such applications requires an external agent to act as intermediary. In this chapter we have proposed the use of J2EE server with its own messaging framework to act as intermediary.

5.2.1 Architecture of the intermediary

Fig. 5.2 : System Architecture

Fig. 5.2 shows the overall system architecture viz. Legacy System Interface (LSI), Data Transformation (DT), Communication Interface (CI), Intermediary software (IS), and J2EE application server. The software components that are shown in dotted rectangle can be configured in following three ways:

a> All software components on J2EE Application server  
b> All software components on legacy system server, if the server is configurable  
c> A new server added in between the legacy system and J2EE application server

These configuration variations are possible since what we are considering is SME and Local area network.

5.2.2. Implementation Issues

All these software components, except some part of LSI are in the form of Enterprise Java Beans (EJBs). LSI uses session beans while DT and CI uses Message Driven Beans (MDBs). The LSI is divided into two parts, first one being specific to legacy system and second being portable. The communication between LSI-DT-CI-IS modules can be synchronous or asynchronous. In case of asynchronous
communication, logs can be maintained between every two components e.g. a log between LSI and DT and another log between DT and CI. The tasks expected by various software components are as follows:

a> **Legacy software:** The legacy system software is modified to certain extent so as to generate triggers/events to initiate first part of LSI on occurrence of local (legacy) database updates.

b> **Legacy System Interface (LSI):** First part receives triggering action from modified legacy software and initiates second part to create communication message which may be handed over directly to DT (in case of synchronous communication) or may be placed in the log provided between the two (in case of asynchronous communication). In reverse scenario it receives message from DT, either directly or through log. These messages are then converted and transferred to legacy software for local (legacy) database updates.

c> **Data Transformation (DT):** It may receive messages from LSI or CI. In first case, upon receipt of message from LSI, the DT converts message into XML format. The XML file may contain DTD or may adhere to standard global DTD. It then hands over the message in the form of XML file to CI for dispatch.

In second case it may receive message from CI in the form of XML file, which it needs to convert into message format, agreed between DT and LSI. The XML format is suggested since using XML, data is more transparent during transfers and facilitates easy adoption of changes to the schemas and business processes [23].

d> **Communication Interface (CI):** The CI is built from a reliable messaging service e.g. Java Messaging Services (JMS). The job of CI on one hand is to accept XML files from DT and convert them into messages, and send them to required destinations, while on the other hand receive messages, detach XML files and hand them over to DT.

e> **Intermediary Software (IS):** This acts as an interface between J2EE server and the portable part of software, e.g. it may be a bean container with its own logs and statistical information. The CI, instead, may maintain logs. These logs are
forwarded to LSI periodically to help LSI decide upon status of transactions/actions triggered by it.

5.2.3 Interface with Legacy System

As mentioned earlier, the LSI is made up of two parts. First part is specific to legacy system and may vary from system to system. This works as interface with legacy system e.g. if the legacy system is payroll and is developed using Foxpro then the interface can have access to *.dbf files of Foxpro directly. A Foxpro dbf file is typically divided into two parts, the first one contains Meta data and the second part contains actual data. Depending upon the version of Foxpro, structure of Meta data varies slightly, which is programmed in LSI and is specific to legacy system installation. In case the legacy system is developed in COBOL (say accounting application), the data files can be directly accessed, upon once knowing the record structure and then programmed into LSI.

However, accessing data files directly do mean disturbing other local functionality. This necessitates “local” agent software on top of legacy or in the form of modified legacy software, which initiates the necessary triggers. The option that is chosen here is to modify legacy software in a way so as to record updates to local database and transaction commits and then triggering first part of LSI. In case of multi-system transactions, which may involve local as well as other legacy system, which divide the transaction into two sub-transactions, one for the local and another for other remote legacy system. The first sub-transaction then can be treated as local transaction, with a difference of commit, which is dependent on successful completion of other sub-transaction. The other sub-transaction is then handed over to DT, converted to XML, handed over to Cl, sent to other legacy system, may be on other server with equal architecture. When successful commit of this sub-transaction is received, the local sub-transaction is also committed. Of course in case of failure/abort of any sub-transaction, the other sub-transaction, and hence the main transaction need to be aborted.
5.3 Experimental Setup

The approach presented in this chapter was based on an experimental setup conducted in an in-house laboratory. There were four computer machines connected through LAN. The platforms/softwares running on them were as follows:

1. Computer 1: A J2EE (1.2) application Server was running under Windows 2K OS. This is intended to be an application server.
2. Computer 2: A database was created in FoxPro 2.6. The system was working under Windows 2K OS. This is intended to be running a Legacy System.
3. Computer 3: A database was created in DB2. The system was working under Windows 2K OS. This is intended to be running a software system developed recently.
4. Computer 4: Oracle Server 8 was running under Windows 2K OS. This is intended to be a Data Warehouse.

The experimental database was considered of a banking application. Only limited transactions viz. Opening an account, deposit to an account, withdraw from an account and money transfer from one account to another were considered. The money transfer transaction was designed to transfer amount from an account at Computer 2 to an account at Computer 3 or visa-versa. FoxPro was chosen for legacy system since FoxPro allows direct access of its database (*.dbf) files. Structural layout of any dbf file is fixed for a particular version of FoxPro. Software design Specification (SDS) of the software is produced below.

5.4 System Architecture Description

5.4.1 Overview of Modules

There are two modules in our EAI software
• Legacy Site module
• Application Server module

Legacy System Module

This module will be concerned with the following work load:

1. Login
2. Firing a transaction.
3. All the transactions are send to Application server module for routing or to be entered into warehouse.
4. Accept and perform sub-transactions routed by application server to this site

Application Server Module

This module will be concerned with the following work load:

1. Transactions are routed to their respective destinations. A log of all such transactions is maintained
2. All committed transactions are loaded to data warehouse.

Legacy System module is further organized into following modules:

1. LS Interface: This module gets the information of transaction form the user JSP pages specific to the enterprise. This information is processed and based on the result the following action will be taken:

   a. If the transaction is local transaction then respective updates is made in local legacy database and its entry is made in DTLog.
   b. If the transaction is remote transaction then respective entry is made in DTLog for sending it to remote site.
   c. If the transaction is distributed transaction containing a local and a remote part then respective updates is made in local legacy
database and entry is made in DTLog for sending it to remote site.

d. If the transaction is distributed transaction containing more than one remote part then respective entries are made in DTLog for sending it to remote site for every remote part.

Also when a transaction arrives from the server the respective transaction is carried out and proper response is generated.

2. DT : This module performs the following two functions:
   a. Reads the DT Log entries made by LS Interface and converts them into XML files and make their entries in CI Log.
   b. Read CI Log entries and XML file created by CI Interface and convert it into general information for which an entry are made in DT Log.

3. CI : This module perform the following two function:
   a. Read the entries in CI log and XML file and transfer the XML file using JMS.
   b. On the reverse when a message arrived at CI, then respective XML file is retrieved from message and entry is made in CI Log.

**Server module is further organized into following modules:**

1. CI : This module receive the message and can perform the following two function:
   a. If the message is meant for Server then respective XML file is retrieved from message and entry is made in CI Log.
   b. If the message is meant for a remote site the proper mapping is done and the message is sent to respective CI.
2. DI: This module work same as DT of Legacy System module, however DT log is called Commit Log at Server module.

3. Server interface: This module do the following work:
   a. Reads the commit log entries.
   b. Check for the proper data type conversion.
   c. Prepare the query to be fire to the database.
   d. Proper response is generated when query is committed to ware house.

5.5 Architecture and Relationships

The architecture and interrelationships of the modules is shown in fig. 5.3
5.6 User Interface Issues

Users of the system are:
- Operator
- Administrator

The operator will be the user who will be firing all the transactions at the legacy system(s) and the one who will be noticing and taking care of all generated responses. Administrators work is to monitor the whole system and to add or remove new sites into the system.

5.7 Detailed Description of Components

5.7.1 Legacy System Module

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is a combination of sub modules running on every Legacy system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>This module is mainly meant for connecting the legacy system with the application in hand and also sending the corresponding entries to the application server. Also accepts and perform sub-transactions routed by application server to this site.</td>
</tr>
<tr>
<td>Function</td>
<td>The main functions performed by this module are:</td>
</tr>
<tr>
<td></td>
<td>• Login Authentication</td>
</tr>
<tr>
<td></td>
<td>• Firing a transaction (specific to application)</td>
</tr>
<tr>
<td></td>
<td>• Generating proper responses depending on the result produced after the transaction is fired.</td>
</tr>
<tr>
<td>Subordinates</td>
<td>Various subordinates present under this module are:</td>
</tr>
<tr>
<td></td>
<td>• Legacy System Interface (LSI)</td>
</tr>
<tr>
<td></td>
<td>• Data Transformer (DT)</td>
</tr>
<tr>
<td></td>
<td>• Communication Interface(CI)</td>
</tr>
</tbody>
</table>
## Dependencies

This module depends on the inputs made by the operator working on the system and on the messages coming from the remote sites for some operation to be carried out.

## Interfaces

The Legacy System Module has only one interface:

- JSP pages for firing transactions specific to the enterprise

## Resources

The module uses all the resources used by individual subordinate modules.

## Processing

The whole processing under this module is carried out by the submodules present. They will process the inputs taken by the JSP pages and then the extracted information is transferred to the application server.

## Data

Data will vary from enterprise to enterprise as for the project scope we are considering banking application, where the data will be the transaction information provided by the operator using JSP pages.

### 5.7.2 Legacy System Interface

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module present at all the legacy systems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>The purpose of this module is:</td>
</tr>
<tr>
<td></td>
<td>• To accept the input data from the JSP pages and pass the required information to the next module for further operations.</td>
</tr>
<tr>
<td></td>
<td>• To perform the operations on the legacy database (If needed).</td>
</tr>
<tr>
<td>Function</td>
<td>The main functions performed by this module are:</td>
</tr>
</tbody>
</table>
After taking all the input data from the JSP pages and then analyze it and accordingly perform the following operations:

1. If the transaction is local transaction then respective updates is made in local legacy database and its entry is made in DTLog.
2. If the transaction is remote transaction then respective entry is made in DTLog.
3. If the transaction is distributed transaction containing a local and a remote part then respective updates is made in local legacy database and entry is made in DTLog.
4. If the transaction is distributed transaction containing more then one remote part then respective entries are made in DTLog.

If any new entry is made into the DTLog by the DT module then the following tasks can be done:
1. If the entry is a transaction then it is performed on the database present.
2. Or if the entry is a response for any previously fired transaction then if the transaction is complete it is written into the DTLog to be sent to the data warehouse.

<table>
<thead>
<tr>
<th>Subordinates</th>
<th>--NA--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependencies</td>
<td>This module depends on the inputs given by the JSP pages and on the entries made into the DTLog by the DT module.</td>
</tr>
<tr>
<td>Interfaces</td>
<td>The Legacy System Interface Module has following interfaces:</td>
</tr>
<tr>
<td></td>
<td>• JSP pages for firing transactions specific to the Enterprise.</td>
</tr>
<tr>
<td></td>
<td>• DT Log, which is the interface between LSI and DT modules.</td>
</tr>
</tbody>
</table>
### Resources

A local meta-data repository for firing local and remote transactions.

### Processing

- Data from the JSP pages is analyzed using the local Metadata repository present on the system. Based on the result, local and distributed transaction is identified. Then the transaction is subdivided if necessary. Entries of all the transactions are made into LSI Log. Local transactions are performed on the Legacy database. For remote sub-transactions entries are made into DT Log.
- Data received from DT module can be a new transaction or response for a transaction.
  - i. If it is a new transaction then its entry is made in LSI Log for transaction to be carried out at local Database.
  - ii. If it’s a response, which indicates the completion of a transaction, then an entry is made in DT Log to be sent to the data warehouse.

### Data

Data will vary from enterprise to enterprise.

### 5.7.3 Data Transformer module

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module present at all the legacy systems.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>The purpose of this module is:</td>
</tr>
<tr>
<td></td>
<td>• Convert the information from DT Log into a standard XML format.</td>
</tr>
<tr>
<td></td>
<td>• Extracting the information from the XML file received from CI module.</td>
</tr>
<tr>
<td>Function</td>
<td>Functions performed by this module are:</td>
</tr>
</tbody>
</table>
- Reads the DT Log entries made by LS Interface and converts them into XML files and make their entries in CI Log.
- Reads CI Log entries made by CI Interface and accordingly read the XML file and extracting the general information about the transaction and putting it into the DT Log.

<table>
<thead>
<tr>
<th>Subordinates</th>
<th>--NA--</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependencies</td>
<td>This module depends on the entries made into the DT Log by the LSI module and the entries made into the CI Log by the CI module.</td>
</tr>
<tr>
<td>Interfaces</td>
<td>The Data Transformer module has following interfaces:</td>
</tr>
<tr>
<td></td>
<td>• DT Log, which is the interface between LSI and DT modules.</td>
</tr>
<tr>
<td></td>
<td>• CI Log, which is the interface between DT and CI modules.</td>
</tr>
<tr>
<td>Resources</td>
<td>A DTD for identifying the format of XML file.</td>
</tr>
<tr>
<td>Processing</td>
<td>It Consist of two parts:</td>
</tr>
<tr>
<td></td>
<td>i. An entry (which is an object of DT Log class) is read from DT Log and Constructing the XML document referring the predefined DTD. Then an entry is made in CI Log.</td>
</tr>
<tr>
<td></td>
<td>ii. An entry (which is an object CI Log class) is read from CI Log and the information from XML file is extracted. An Object of DT Log class is created and saved to DT Log.</td>
</tr>
<tr>
<td>Data</td>
<td>Data consist of Objects of DT Log class which is used for storing entries in DT Log and on the other hand the XML file.</td>
</tr>
<tr>
<td></td>
<td>• Format of DT Log class:</td>
</tr>
</tbody>
</table>
public class DTLog
{
    char Transaction;
    String Source_siteID;
    String Transaction_no;
    String Transaction_type;
    Relevant_info r_i;
    String Response=null;
    String Destination_siteID;
    char Transmission;
    String Read;
    String For;
}

public class ReadDTLog()
{ /* This Function is used to read an object from DT Log. */
}

Public WriteDTLog()
{ /* This Function is used to write an object into DT Log. */
}

public class Relevant_info
{
    String attrib_name[];
    String attrib_value[];
}
### 5.7.4 Communication Interface Module

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module present at all the legacy systems.</th>
</tr>
</thead>
</table>
| Purpose | The purpose of this module is:  
- Communication between Legacy System module and Application Server module. |
| Function | Functions performed by this module are:  
- Routing of XML file to Application Server CI.  
- Receiving XML files from application server module and handing it over to DT module by writing its entry in CI log. |
| Subordinates | --NA-- |
| Dependencies | This module depends on the inputs provided by the DT module through CI log, and on the other side on the inputs from the CI module at the Application Server side. |
| Interfaces | The Data Transformer module has following interfaces:  
- CI Log, which is the interface between DT and CI modules.  
- Connection to Application Server CI via JMS. |
| Resources | --NA-- |
| Processing | It processes in two parts:  
- Read new entries from CI Log and will send XML files to CI module at Application Server side along with some header information.  
- When any message comes at the CI module its entry is made into the CI log and the XML file is handed over to DT module. |
| Data | Data consist of Objects of CI Log (a class which is used for |
storing entries in CI Log) and the respective XML files.

Format of CI Log Class:

```java
Class CILog {
    String destination_siteID;
    String source_siteID;
    String filename;
    String for;
    String read;
    String error;

    Public ReadCILog() 
    {/* This Function is used to read an object from CI Log. */}

    Public WriteCILog() 
    {/* This Function is used to write an object into CI Log. */}
}
```

5.8 Application Server Module

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is a combination of sub modules running on every Application Server system.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>The main purpose of this module consists of two parts:</td>
</tr>
<tr>
<td></td>
<td>• Act as router, to route messages between different Legacy System.</td>
</tr>
<tr>
<td></td>
<td>• To make entries of all committed transaction into data warehouse.</td>
</tr>
<tr>
<td>Function</td>
<td>The main functions performed by this module are:</td>
</tr>
<tr>
<td></td>
<td>• Accept messages from various Legacy sites and analyze it</td>
</tr>
</tbody>
</table>
1. If the message is meant for any other legacy site, then it is routed to the proper destination.
2. If the message is meant to be commitment in data warehouse then the proper update is made.

### Subordinates

Various subordinates present under this module are:
- Communication Interface (CI)
- Data Transformer (DT)
- DWH Interface.

### Dependencies

Depends on the inputs from Legacy System modules.

### Interfaces

No user interface, however it has a JDBC connectivity with the DWH. It is connected to all other Legacy System module through JMS.

### Resources

A metadata repository is used containing metadata for data of all Legacy sites.

### Processing

The whole processing under this module is carried out by the submodules present. Accepting messages from various Legacy System modules, routing of messages and commitment of transactions to data warehouse.

### Data

Data consist of the JMS messages from different legacy system module containing information about various transactions occurred.

---

**5.8.1 Communication Interface Module**

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>The purpose of this module is:</td>
</tr>
<tr>
<td></td>
<td>• Communication between Legacy System modules and Application Server module.</td>
</tr>
<tr>
<td>Function</td>
<td>Functions performed by this module are:</td>
</tr>
<tr>
<td>----------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td></td>
<td>• Receiving XML files from Legacy System module and routing it appropriate other Legacy System module.</td>
</tr>
<tr>
<td></td>
<td>• Receiving XML files from Legacy System module and handing it over to DT module by writing its entry in CI log.</td>
</tr>
</tbody>
</table>

| Subordinates | --NA-- |

| Dependencies | This module depends on the inputs provided by the DT module through CI Log, and on the other side on the inputs from the CI module at the Legacy System module. |

<table>
<thead>
<tr>
<th>Interfaces</th>
<th>The Data Transformer module has following interfaces:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• CI Log, which is the interface between DT and CI modules.</td>
</tr>
<tr>
<td></td>
<td>• Connection to Legacy System CI via JMS.</td>
</tr>
</tbody>
</table>

| Resources | --NA-- |

<table>
<thead>
<tr>
<th>Processing</th>
<th>It processes in two parts:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• When any message comes at the CI module</td>
</tr>
<tr>
<td></td>
<td>1. Its entry is made into the CI log and the XML file is handed over to DT module.</td>
</tr>
<tr>
<td></td>
<td>2. Or routed to respective Legacy System module.</td>
</tr>
<tr>
<td></td>
<td>• Read new entries from CI Log and will send XML files to CI module at Legacy System module along with some header information.</td>
</tr>
</tbody>
</table>

| Data | Data consist of Objects of CI Log (a class which is used for storing entries in CI Log) and the respective XML files. Format of CI Log Class is same as in each Legacy System module mentioned above. |
### Data Transformer Module

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module present at all the legacy systems.</th>
</tr>
</thead>
</table>
| Purpose | The purpose of this module is:  
- Convert the information from Commit Log into a standard XML format.  
- Extracting the information from the XML file received from CI module. |
| Function | Functions performed by this module are:  
- Reads the Commit Log entries made by DWH Interface and converts them into XML files and make their entries in CI Log.  
- Reads CI Log entries made by CI Interface and accordingly read the XML file and extracting the general information about the transaction and putting it into the Commit Log. |
| Subordinates | --NA-- |
| Dependencies | This module depends on the entries made into the Commit Log by the DWH interface and the entries made into the CI Log by the CI module. |
| Interfaces | The Data Transformer module has following interfaces:  
- Commit Log, which is the interface between DWH interface and DT modules.  
- CI Log, which is the interface between DT and CI modules. |
| Resources | A DTD for identifying the format of XML file. |
| Processing | It consist of two parts:  
iii. An entry (which is an object of Commit Log class) is
read from Commit Log and Constructing the XML
document referring the predefined DTD. Then an
entry is made in CI Log.

iv. An entry (which is an object CI Log class) is read
from CI Log and the information from XML file is
extracted. An Object of Commit Log class is created
and saved to Commit Log.

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data consist of Objects of Commit Log class which is used for storing entries in Commit Log and on the other hand the XML file.</td>
</tr>
<tr>
<td>• Format of Commit Log class:</td>
</tr>
</tbody>
</table>

```java
public class CommitLog {
    String Source_siteID;
    String Transaction_no;
    String Transaction_type;
    RelevantInfo r_i;
    String Response=null;
    String Destination_siteID;
    String Read;
    String For;
    Public ReadDTLog() { /* This Function is used to read an object from DT Log. */
    }
    Public WriteDTLog() { /* This Function is used to write an object into DT Log. */
    }
}
```

```java
public class Relevant_info {
}
```
### 5.8.3 Data Warehouse (DWH) Interface

<table>
<thead>
<tr>
<th>Type</th>
<th>This module is subordinate of the above module present at all the legacy systems.</th>
</tr>
</thead>
</table>
| Purpose | The main purpose of this module is:  
- Commitment of transactions to data warehouse. |
| Function | Functions performed by this module are:  
- Reads the Commit Log entries convert them into proper queries to be fired to data warehouse.  
- Proper data format conversion is done is necessary. |
| Subordinates | --NA-- |
| Dependencies | This module depends on the entries made into the Commit Log by the DT module. |
| Interfaces | The Data Transformer module has following interfaces:  
- Commit Log, which is the interface between DWH interface and DT module. |
| Resources | A meta data repository is used for mapping and Data type conversion. |
| Processing | It Consist of two parts:  
I. An entry (which is an object of Commit Log class) is read from Commit Log, Proper mapping of fields and data type conversion is done. |
### II. An entry (which is an object Cl Log class) is made in Server Log and a query is formatted to firing the transaction to data warehouse.

| Data | Data consist of Objects of Commit Log class which is used for storing entries in Commit Log containing information of transaction. |

---

### 5.9 Reuse and Relationship to Other Products:

We have designed our software modules with adequate foresight so that whatever we implement can finally be extended and implemented later on a larger scale with added features or modules can be taken away and attached to some other architecture. These are some of the ways in which the software modules can be reused:

- The EAI software can be used for varied number of Enterprise application with some application specific changes. The architecture is so build that it can be used for varied application efficiently.

- The Legacy System Module is designed in such a way that any new branch can be easily introduced to the existing Enterprise (which is running with EAI). All the classes used in Legacy System Module are reusable as our software provides the facility of using Enterprise Beans hence, will be less complex to do above task.

- In future, queries can be fired to data warehouse directly.

### 5.10 Design Decisions and Tradeoffs

- For building the EAI software we have used EJBs for handling business logic. EJBs are used for large-scale project as they are portable components, the application assembler can build new applications from existing beans.
• Many operators can work simultaneously as we are using JSP as user interface. JSP provides an efficient way to display dynamic content.

• Totally autonomous software – Operator comes into picture only once when data is fed for the transaction.

• Multiple transactions can be fired from various Legacy Systems. Concurrent transactions are also handled by Application Server as we are using EJB technology.

5.11 Pseudo Code for Components

• Calling of an session bean using JSP

```java
try {
    InitialContext ic = new InitialContext();
    Object objRef = ic.lookup("java:comp/env/ejb/selfbeanwar");
    MainHome home = (MainHome)PortableRemoteObject.narrow(objRef, MainHome.class);
    mr = home.create();
}
```

• Connectivity with local legacy database:

```java
public void makeConnection()
{
    try
    {
        //InitialContext ic = new InitialContext();
        //DataSource ds = (DataSource) ic.lookup(dbName);
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
}
```
con=DriverManager.getConnection("jdbc:odbc:DSN_Bank");
//DSN - Data Source Name
//con = ds.getConnection();
st=con.createStatement();

try {
    Context jndiContext = new InitialContext();
    queueConnectionFactory = (QueueConnectionFactory)
        jndiContext.lookup("QueueConnectionFactory");
}

• **Preparation of XML file using information from DT Log**

    FIS = new FileInputStream("xml.dtd");
    Parser parser = new Parser("xml.dtd");
    DTD dtd = parser.readDTDStream(FIS);
    Enumeration en = dtd.internalElements();

    TXDocument doc = new TXDocument();
    ele = (ElementDecl)en.nextElement();
    root = (TXElement)doc.createElement(ele.getName());
    .
    .
    doc/printWithFormat(wr);

• **JMS messaging:**

    try {
        Context jndiContext = new InitialContext();
        queueConnectionFactory = (QueueConnectionFactory)
            jndiContext.lookup("QueueConnectionFactory");
    }
\begin{verbatim}

ringQueue = (Queue)
jndiContext.lookup("RingQueue"); // queue name
}

try {
    queueConnection = queueConnectionFactory.createQueueConnection();
    queueSession = queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
    queueSender = queueSession.createSender(ringQueue);
    textMessage = queueSession.createTextMessage();
    queueSender.send(textMessage);
}

\end{verbatim}

- **Oracle Data Warehouse Thin Driver connectivity:**

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

Connection conn =

DriverManager.getConnection("jdbc:oracle:thin:@mml5:1521:eai","scott","tiger");
5.12 Summary of Work

In this chapter, we have presented our work in the form of a simple approach for integration of applications that run on a LAN. Typical application setup considered is of a SME wherein there are one or more servers, may be Linux/Windows based, and number of independently working modules which are typical legacy systems. Keeping in view the cost factor, the architecture emphasizes use of all open source software components like J2EE, XML, and JMS. Considered geographically, servers are not at far distance. Due to LAN environment, reliable messaging is possible and also the distributed transaction commit protocol related issues are simpler. The DT, CI and part of LSI are totally portable and only part of LSI acting as interface to legacy system is specific and needs to be rewritten/rebuild, though once for a specific version/ platform. Owing to all these factors, it is easy to conclude that such an approach for integration is affordable to SMEs.

A paper based on this work, titled “Application Integration Strategy for SME Segment in Developing Countries” has been submitted for international conference namely “European Management and Technology Conference”, Rome, Italy, 20-21 June 2005. (emt@triof.org).

This paper has been accepted [56] for presentation and publication in the conference proceedings.