Chapter 6

Implementation

6.1. Introduction

Last chapter is for the analysis and design. It provides all the necessary artifacts for implementation. This chapter will discuss about the implementation methodologies I carry out in implementing the system.

6.2. SAP Java connection

One of the important implementation step is setting up SAP Java connection. For that I use the SAP Java Connector (SAP JCo), a middleware component that enables the development of SAP-compatible components and applications in Java. I use the JCo 2.0 version with JDK 1.5.

6.2.1. JCo implementation

The instructions for the installation of SAP JCo are as follows. Below instructions are performed in local server.

Follow the steps below.

- Create a directory, for example C:\\SAPJCo, and extract the JCo .zip file into this directory.
- Copy the file librfc32.dll from the SAP JCo main directory to C:\WINNT\SYSTEM32, as long as the version that is already there is not a more recent version than the one that is delivered with the SAP JCo.
- Make sure that the file sapjco.jar (in the SAP JCo main directory) is contained in the *class path* for all projects for which you want to use the SAP JCo..

For productive operation, the following files from the SAP JCo .zip file are necessary:

1. sapjco.jar
2. librfc32.dll
3. sapjcorfc.dll

6.2.2. JCo connection

To open a connection with SAP server following Java code is used.

```
import com.sap.mw.jco.*;

private JCO.Client sapClient;
private JCO.Repository repository;
private IFunctionTemplate functionTemplate;
private JCO.Function remoteFunction;
```



```
sapClient =   JCO.createClient("400", // SAP client
                  "usernam",                    // userid
                 "password",                   // password
                  null,                          //  log on language
               //    "10.226.200.XXX",              // SAP server host name
                 "/H/220.247.208.22/S/3299/H/", //or router string
                 "50"); // system number
   sapClient.connect();
   System.out.println(sapClient.getAttributes());
```

Now the sapClient object can be use to call SAP functions from Java. In order to being called SAP functions, they should be remote enable. ZUPLOAD_ZLOCNF is a SAP function that is remote enable and will receive plant,Id, module,UPC id, qty.

In the end of successful execution the connection has to be disconnected.

```
//Configure SAP RFM used.
repository = new JCO.Repository("RFM Repository", sapClient);
functionTemplate = repository.getFunctionTemplate("ZUPLOAD_ZLOCNF");
remoteFunction = functionTemplate.getFunction();
   }
```

Passing parameters to function “ZUPLOAD_ZLOCNF”:

```
   public int upload(int plant, String id, String module_id, String upc, int quantity){
            //Import parameters
JCO.ParameterList imPara = remoteFunction.getImportParameterList();
imPara.setValue(plant,     "IM_PLANT");
imPara.setValue(id,        "IM_ID");
imPara.setValue(module_id, "IM_MODULE_ID");
imPara.setValue(upc,       "IM_UPC");
imPara.setValue(quantity,  "IM_QUANTITY");

//Execution
sapClient.execute(remoteFunction);

//Export parameters- Returns the upload confirmation
JCO.ParameterList exPara = remoteFunction.getExportParameterList();
int sy_subrc = exPara.getInt("EX_IS_SAVED");
            return sy_subrc;
   }
```

6.3. IP scanner connection

Below portion of the code will describe how to connect to IP scanners via java. I have used the java sockets.

Server socket name is : providerSocket

- **Initialization**

```
providerSocket = new ServerSocket(2004, 20);    //port no, number of  conncection
System.out.println("ServerSocket configured.");
```

- **Assignment**

```
Socket socket = null;
InputStream in = null;
String ipAddress = "";
socket = providerSocket.accept();
ipAddress = socket.getInetAddress().getHostAddress();
System.out.println("IP :"+ipAddress+" connected");
```


- **Read input stream**

```
in = socket.getInputStream();                    // Buffered input stream
```

- **Update to local database by DBentry class**

```
int RETURN = 13;                          //check the Enter key
String scanned_code = "";
int byteVal = 0;
int count = 1;
        do{
            byteVal = in.read();
                                          //Save to the local database server
          if(byteVal==RETURN){            // if Enter key  received
              new DBEntry(ipAddress,scanned_code,count);   //local table update
              scanned_code = "";
              count++;
```

```
        }
        else{
            scanned_code = scanned_code + (char)byteVal;
        }
    }while(true);
```

6.4. SQL Table updates

To write the data to sql tables I use following

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

con = DriverManager.getConnection("jdbc:odbc:ds1");
String sql = "INSERT INTO TABLE_NAME  (ip,upc) VALUES(?,?)";

PreparedStatement stmt = con.prepareStatement(sql);
stmt.setString(1,this.ip);
stmt.setString(2,this.upc);
stmt.executeUpdate();
```


6.5. Passing messages to display boards

Display board update is implemented using the following steps

- **Prepare the message**

```
String board = "<ID"+boardNo+">";
String page = "<P01>";
String msg = board + page + "<CP><FD>Module Output:" + count+ "\r\n";
```

- **Open the port connection**

```
String COMM_PORT_NAME = "COM1";
CommPortIdentifier portId;
```

```
SerialPort port;
OutputStream outputStream;

// parse ports and if the default port is found, initialized the reader
Enumeration portList = CommPortIdentifier.getPortIdentifiers();
boolean portFound = false;
while (portList.hasMoreElements()) {
        portId = (CommPortIdentifier) portList.nextElement();
        if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
                if (portId.getName().equals(COMM_PORT_NAME)) {
                        System.out.println("Found port: COM1");
                        port = (SerialPort) portId.open("Display Board
Application", 2000);
                        //set port parameters
                                port.setSerialPortParams(
                                                9600,
                                                SerialPort.DATABITS_8,
                                                SerialPort.STOPBITS_1,
                                                SerialPort.PARITY_NONE    );

                        //get the outputstream
                        outputStream = port.getOutputStream();
                        port.notifyOnOutputEmpty(true);

                        System.out.println("Port: COM1 Configured.");

                        portFound = true;
                }
        }
}
```

- **Pass the message to the stream**

```
//write string to serial port
outputStream.write(msg.getBytes());
System.out.println("Message Sent.");
```

6.6. Get the time stamp to track table updates

Time stamp is used in saving the Line Out scanned records separately to Log tables. It can be use to separately identify each of the Line Out scan in same UPC sticker in same module.

```
import java.util.*;
import java.text.SimpleDateFormat;

String mytime = now("yyyyMMdd");
        String mydate = now("hhmmss");
```



```
// now- method declaration
String now(String dateformat) {
        Calendar cal = Calendar.getInstance();
     SimpleDateFormat sdf = new SimpleDateFormat(dateformat);
    return sdf.format(cal.getTime());
  }
```

6.7. Summary.

I have discussed the implementation methods used in Java developments to perform various tasks in above chapter. Then the new chapter will be discussing the evaluation techniques of the development.