Chapter 2: Literature Review

2.1 Introduction

In order to start to do the dissertation, a deep study regarding the data automation of Microsoft Excel Application has been carried out. As a whole, this chapter will be divided into two major parts, which are the usage of Microsoft Excel and the development of Java program to automate Excel.

Furthermore, aspects such as Microsoft Excel file format, Java language and its usage in calling Components Object Model (COM) provided by Microsoft will also be explained and elaborated in this chapter.

2.2 The usage of Microsoft Excel Application

Microsoft Excel dominates the spreadsheet market. Not too long ago, Lotus 1-2-3 was considered the "standard" spreadsheet. However, Excel now holds that distinction, with an approximate 90% market share (Microsoft Excel – The Spreadsheet Page, 2003).

Microsoft Excel started to be used in the year of 1987. It was originally developed for Macintosh. The first Excel version was labelled "2" to correspond to the Mac version. It was one of the first spreadsheets to use a graphical interface with pull down menus and a point of click capability using a mouse-pointing device. Due to the advent of computing

technologies, Microsoft Excel is now being expanded and equipped with multi-sheet workbooks functionality.

Today, the latest version of Excel which is known as Excel 2003, has a long list of new features. The most significant feature is the ability to recover the work when Excel crashes.

2.2.1 The Usage Area

The main contribution of Microsoft Excel is in handling statistical jobs. For its facilities in tabulations, pivots tables and graphical presentation, many statisticians have chosen to use Excel rather than using any other statistical software. Excel has also proven its functionality in providing dynamic nature and power of data manipulation.

Besides that, different types of business organisations have also been using Microsoft Excel as a tool for their operational business analysis. For example, accounting firms use Excel to record their account spreadsheet in electronic form. The spreadsheet shows all of the costs, income, taxes, etc on a workbook for a manager to look at when making decisions.

Excel can be used to organize information into columns and rows. The data can then be added up by a formula to give a total or sum. Furthermore, it summarizes information from many paper sources in one place and presents the information in a format to help a decision maker see the financial "big picture" for the company.

2.3 A look into Excel file format

Microsoft Excel uses a file format called Binary File Format (BIFF). There are many types of BIFF records. Each has a 4 byte header. The first two bytes make up an opcode that specifies the record type. The next two bytes specify record length. Header values are stored in byte-reversed form (less significant byte first). The rest of the record is the data itself. Figure 2.1 illustrates the file format of the header.

	Record Header	Record Body
Byte number	0 1 2 3	0 1 ...
Record Contents	XX | XX | XX | XX	XX | XX | ...
	Opcode | Length	Data

Figure 2.1 Excel File Format

2.4 Java Technology

In order to develop an object-oriented data automation system, the Java technology, which is both a programming language and a platform, has been studied. The code that is written in Java and run directly on the Java platform is a standalone program. This standalone program can be executed in any machine that has installed with Java Virtual Machine (JVM).

2.4.1 The Java Programming Language

The Java programming language is a high-level language that can be characterized as object-oriented, distributed, interpreted, portable, robust and multithreaded.

The Java programming language is designed to be object oriented from the ground up. Programmers using the language can access existing libraries of tested objects that provide functionality ranging from basic data types through I/O and network interface to graphical user interface toolkits. These libraries can be extended to provide new behaviour.

With most programming languages, the program must be either compiled or interpreted so that it can be executed on a machine. The Java programming language is unusual in the sense that a program is both compiled and interpreted. First, the Java program will be translated into an intermediate language called Java bytecodes – the platform-independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java bytecode instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.

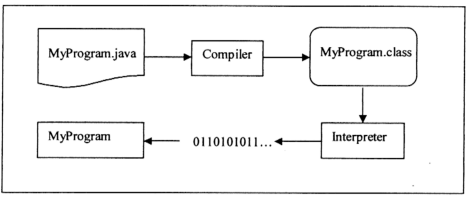

Figure 2.2 Execution of a Java program

The java bytecodes help make "write once, run anywhere" possible. A java program can be compiled into bytecodes on any platform that has a Java compiler. The bytecodes can then be run on any implementation of the Java Virtual Machine (JVM). This means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.

2.4.2 The Java Platform

As the Java technology consists of programming language and a platform, a description regarding the Java platform will be presented here.

A platform is the hardware or software environment in which a program runs. Windows XP, Linux, Solaris and MacOS are some of the most popular platforms. Normally, most of the platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java platform has two components, namely the Java Virtual Machine (Java VM) and the Java Application Programming Interface (Java API). Java VM is the based for the Java platform and is ported onto various hardware-based platforms. It is responsible for interpreting all the java classes that have been compiled.

The Java API is a large collection of ready-made components that provide many useful capabilities, such as graphical user interface (GUI) wizards. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. Among the area of facilities provided by Java API are stated as below:

i. The essentials: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.

ii. Applets: the set of conventions used by applets

iii. Networking: URLs, Transmission Control Protocol (TCP), User Datagram Protocol (UDP) sockets and Internet Protocol (IP) addresses.

iv. Internationalization: the Java API provides a mean for writing programs that can be localized for users worldwide. With this, programs can automatically adapt to specific locales and be displayed in the appropriate language.

v. Security: encryptions, digital signatures, session authentication and private as well as public key management provide different levels of security in a Java program.

vi. Software components: the JavaBeans is a type of component that can be plug into existing system architectures.

vii. Java Database Connectivity (JDBC): provides uniform access to a wide range of relational databases such as IMB DB2, Oracle, Microsoft SQL and Informix.

2.4.3 Applying Java Technology in Building Object-oriented Data Automation System

The Java technology, both the programming language as well as the platform provide a better way of doing programming. Firstly, it promotes writing less code. A Java class can be reused and deployed in any project that is similar in its functional aspects. Secondly, the Java technology encourages good coding practices. It provides garbage collection that help in avoiding memory leaks. Besides that, its extensible API allows programmers to reuse other people's tested code and introduce fewer bugs.

"Write once, run anywhere" is another factor that leads the author to use Java technology. Any Java program can be run consistently on any platform because it is compiled into machine-independent bytecodes. Furthermore, the ultimate output of the Java program, can be distributed and accessed easily in any machine without recompiling the entire Java program.

2.5 Concept of Interfacing

In order to interface with any objects that are constructed from different programming languages, a common standard has to be set for describing the objects. Nowadays, there are two standards that are widely used for describing objects. The standards are Java Native Interface (JNI) developed by Sun Microsystems and CORBA Interface Definition Language (IDL) developed by Object Management Group (OMG).

Platform independence is the key factor that leads to the development of a common standard for describing objects. An object developed in C++ should be able to be used in other programming environment such as Java without modifying the entire C++ object source code. This feature will contribute to the flexibility of programming. Besides that it has also promotes object reusability and speeds up the system development process. Further information regarding Object Interfacing could be obtained from Command CORBA® Side by Side: Architectures, Strategies & Implementation (Jason, 1999) and Java Native Interface (Rob, 1998).

2.5.1 Java Native Interface (JNI)

JNI is the link between a Java application and native code. It allows a java programmer to invoke native methods from an application or applet, pass arguments to these methods, and use the results returned by the methods. This native concept includes any methods that would have been declared in other programming languages.

To explain the JNI in more details, a simple example that invokes a native method coded in C++ is shown. Figure 2.3 shows a simple Java code that loads a DLL called nevnative and invokes a method called display().

```
public class NevNative {

public native void display();

static {
System.loadLibrary("nevnative");
}

public static void main(String[] args) {
NevNative object=new NevNative();
object.display();
}
}
```

Figure 2.3 NevNative.java

The system.loadLibrary function is responsible for loading the DLL library and it has to be placed in the static block. This is to ensure that once the instance of this NevNative object is created, the DLL will be automatically loaded in the Java program. After writing the NevNative.java code, it should be compiled by using a Java command, as shown in Figure 2.4.

```
C:\jdk\bin\javac workingdir\NevNative.java
```

Figure 2.4 Compile NevNative.java

After a successful compilation of the Java file, a C header file must be created. A java command called javah is used to do so.

```
C:\jdk\bin\javah workingdir\NevNative
```

Figure 2.5 Create a C Header File

Then, check the working directory and a file called NevNative.h should be found. Open the file and there is a line stated as in Figure 2.6.

```
JNIEXPORT void JNICALL Java_NevNative_display(JNIEnv *, jobject);
```

Figure 2.6 The NevNative.h File

The line shown in Figure 2.6 is the method declaration for the display method which will have to be implemented in the C file.

For the actual C file, a method called display has to be written in order for the NevNative.java program to invoke it. A file called NevNativeImp.c is created and it is shown in Figure 2.7.

```
#include <jni.h>
#include "NevNative.h"
#include <stdio.h>

JNIEXPORT void JNICALL Java_NevNative_display(JNIEnv *env, jobject
obj)
{
printf("Hello world!\n");
return;
}
```

Figure 2.7 The NevNativeImp.c File

The NevNativeImp.c is a simple C code. For method declaration, it should be exactly same as the line stated in the NevNative.h (Figure 2.6). The display() method is meant for printing a line of simple message. Two header files which are the NevNative.h and the jni.h must be included to avoid compilation error. The next step would be compile the C file into a DLL file called nevnative.dll.

In order to verify the implementation of JNI, the NevNative class should be executed by using a java command as shown in Figure 2.8.

```
C:\jdk\bin\java workingdir\NevNative
```

Figure 2.8 Executing NevNative Class

If the message "HelloWorld" is displayed on the screen, then it proves that the JNI implementation has succeeded. JNI has been used for enabling two different programming languages to talk to each other.

2.5.2 CORBA IDL

The Common Object Request Broker Architecture (CORBA) is a standards-based distributed computing model for object-oriented applications developed by the Object Management Group (OMG), a group of 700 vendors and user members including HP, Novell, Sun, IBM, and Digital.

CORBA is an architecture which uses an Object Request Broker (ORB) to send requests from objects executing on one system to objects executing on another system. The ORB allows objects to interact in a heterogeneous, distributed environment, independent of the computer platforms on which the various objects reside and the languages used to implement them. For example, a C++ object running on one machine can communicate with an object on another machine which is implemented in Common Lisp.

A CORBA-compliant application comprises a client and a server. The client is responsible for invoking operations on objects which are managed by the server, and the server receives invocations on the objects it manages and replies to these requests. A Common Lisp object, can either use CORBA services available over the network (as a client), or it can publish services to other components in the application (as a server). The Object Request Broker (ORB) manages the communications between client and server using the Internet Inter-ORB Protocol (IIOP), which is a protocol layer above TCP/IP.

A distributed CORBA application provides a communication medium to any objects regardless of the language in which they are implemented. This can be achieved with the ORB through an interface written in the CORBA Interface Definition Language (IDL). The IDL is included in CORBA version 2.0 specification. It is designed to describe the interface of objects, including the operations that may be performed on the objects and the parameters of those operations. The behaviour of an object is thus captured in the interface independently of the object's implementation. Clients need to know an object's interface in order to make a request. Servers will respond to requests made on those

interfaces, and clients do not need to know the actual details of the server's implementation.

To implement an interface, CORBA IDL is compiled into the source code language with which the client or server is implemented. On the client side, this code is called a stub. On the server-side, this IDL code is called a skeleton.

In order to request a service from the server, the client application calls the methods in the stub (moving down the protocol stack from the client to the stub and then to the ORB). Requests are then handled by the ORB, and enter the server via the skeleton (moving up the protocol stack: i.e. an upcall). The server object then serves the request and returns the result to the client. Figure 2.9 shows the communication between a client and a server by implementing CORBA.

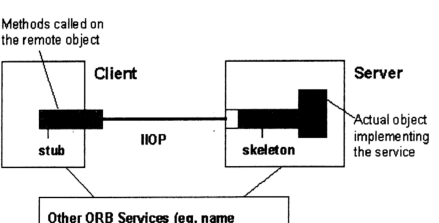

Figure 2.9 Client-Server Communication using CORBA

2.6 Calling COM objects provided by Microsoft using Java Technology

Component Object Model (COM) defines a language-independent binary standard for component interoperability. Microsoft has defined COM interfaces that provide common functionality.

Therefore, in order to access and communicate with the entire Excel objects provided in the Excel object library file (namely excel8.olb), it is necessary to first study what is COM.

2.6.1 Overview of Component Object Model (COM)

The Object Linking Embedding (OLE) was developed before the development of COM. COM is now widely deployed because each COM component can potentially implement several standard or custom interfaces to expose its functionality. These interfaces are the binary standards through which clients and component objects communicate.

Basically, every COM interface and class has its own globally unique identifier (GUID). The GUID is a 128-bit identification number that uniquely represents the COM interface or class across platforms, computers and applications.

COM interfaces can be defined with the Interface Definition Language (IDL). These definitions can be converted into binary form by the Microsoft Interface Definition Language (MIDL) compiler.

The life cycle of a COM object starts with an invocation to a COM object where an object is returned in an IUnknown pointer form. An interface layer called QueryInterface is used to request the interface that is needed. Then, the QueryInterface relases the IUnknown pointer. This pointer is used to execute methods/functions. Finally, the pointer is released as soon as it is no longer needed.

2.6.2 Purpose of Calling COM Objects

What is the purpose of calling a COM object? The COM object is a Microsoft specific protocol for inter-process communication. In order to access and automate all the functions/methods provided in Excel application, it is important to be able to communicate with COM objects. However, COM only works on Windows platform. Microsoft does provide a way of communicating with COM objects by using a COM Wrapper.

The COM Wrapper is responsible to create Java classes that act as the interface between a Java program and the COM objects needed. The main concern in this project is the Excel COM objects. Nevertheless, this COM Wrapper can only be used firmly under Microsoft Java Virtual Machine (Microsoft JVM).

In order to achieve the objective of creating a platform-independent data automation system, it is necessary to use a Java-COM bridge that links between the Java technology and COM.

2.6.3 The JACOB Project – A Java-COM Bridge

With the extensive use of Microsoft Excel as a business tool, many Java based e-business applications and back-end systems need to share information and interact with Microsoft Excel application. At present, there are a few Java based projects that has been carried out in order to develop bridges that link between Java programs and the Microsoft Excel COM objects. Table 2.1 describes five providers that are currently available in the development of Java-COM Bridge.

Table 2.1 The Java-COM Bridge Providers

No.	Java-COM Bridge Provider	Description	Open-Source
1	Intrinsyc's J-Integra Suite	J-Integra is a bi-directional pure Java-COM bridge. Finance, IT, pharmaceutical and defense companies are using J-Integra in an enormous number of diverse ways, such as accessing MS Excel from Java, accessing Enterprise JavaBeans from Visual Basic, and accessing COM objects from JavaServer Pages.	No
2	IBM's Bridge2Java	Interface Tool for Java is a tool that allows Java programs to communicate with ActiveX objects. It allows easy integration of ActiveX objects into a Java Environment. Using the Java Native Interface and COM technology, Interface Tool for Java allows an ActiveX object to be treated just like a Java object.	No
3	infoZoom's jacoZoom	jacoZoom is a collection of java-packages, which allows the users to automate COM/Automation-Servers using java. It is based on JNI and thus can be used with any java environment on the MS Windows platform. It contains a commandline-utility, which produces java wrapper classses directly corresponding to the COM-objects and interfaces.	No
4	Neva Object Technology' s Java2COM	Java2COM is a bi-directional Java-COM bridging tool that enables Java applications to use COM objects and makes possible to expose Java objects as if they were COM objects. Java2COM, which was designed to	No

		provide lightweight wrappers for COM protocols, attempts to preserve various semantics of COM. The main benefit of such an approach is two-folded: on one hand, it simplifies porting of existing Visual Basic or C++ code to Java and gives COM-savvy developers a head-start. On the other hand, it makes it easier to keep up with changes in COM technology while COM evolves.	
5	Jacob Project's JACOB	JAVA-COM Bridge that can be used to call COM Automation components from Java. It uses JNI to make native calls into the COM and Win32 libraries. It's an open source project and it provides freedom for application developers to modify and enhance the Java objects to suite different kinds of requirements.	Yes

A comparative study has been carried out in order to choose which of the available Java-COM Bridge providers for developing this project. There are a lot of similarities among these Java-COM Bridge providers. Basically, all the Java-COM Bridge providers provide java classes for developers to use and hence shorten system development life cycle.

Here, the JACOB project is selected as the tool for calling COM automation for Microsoft Excel application. The main reason for using the JACOB is that it provides sufficient resources, guidelines and at the same time it allows system developers to modify the source-code. Hence, the existing classes can be reprogrammed and refined for better system performance. The remaining Java-COM Bridge providers do not expose their source codes. This has resulted limitation in modifying and enhancing classes.

JACOB is a Java-COM bridge that allows Java programmer to call COM Automation components from Java. It uses Java Naming Interface (JNI) to make native calls into the COM and Win32 libraries. The JACOB project started in the year of 1999 and is being actively used by thousands of developers worldwide (The Jacob Project, 2002).

Due to the availability of the JACOB's source codes, many system developers has already benefited from it because any programmer can make modifications to the code and submitted them back for the enhancement of the project. Most importantly, it provides a means of interaction with essential Excel objects and collections, such as Application, Workbook, Worksheet and Range. The latest version of JACOB project is version 1.7.

The JACOB version 1.7 works with any Java Virtual Machine, including Microsoft Java VM, and Sun Java VM. The JACOB binary distribution includes a Java JAR file and a dynamic link library (DLL) file. Below is the instruction on how to leverage the usage of JACOB in calling COM objects for Microsoft Excel application:

i. jacob.jar: a JAR file for the java classes which must be added to the JAVA CLASSPATH. By using this jar file, all COM objects (which belong to Microsoft Excel application) will have to be renamed to com.ms.jacob.com.*. For example com.ms.com.Variant (used in Microsoft Excel application) should be replaced to com.ms.jacob.com.Variant.

ii. jacob.dll: a small Win32DLL which must be added to the operating system path. In the Microsoft Windows platform, the operating system path is located at C:\Windows\system.

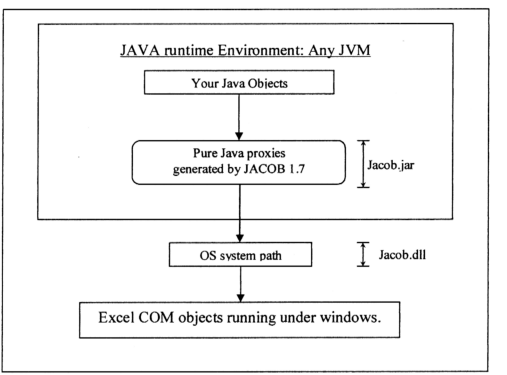

Figure 2.10 Using JACOB to call COM objects

2.6.3.1 Usage and Documentation

The JACOB project provides com.jacob classes that are intended to be compatible with the Microsoft com.ms classes. Therefore, it is important to have the documentation of Microsoft's System Development Kit (SDK) for Java. The documentation of Microsoft's SDK for java provides a good reference and descriptions which explains the methods/properties of each of the Excel COM objects.

2.7 Similar Projects – Microsoft Office Automation using C++, J++ and Visual Basic

No doubt, a successful project should be inspired by successful and informative projects, which were developed previously. Hence, three similar projects that were carried out before are used as references for developing this project. The projects are listed as below:

- An automation program on Microsoft Word files that is developed by Poonam Baijaj (Word Automation C++ Class, 2002).
- A Visual Basic program created by Richard R. Taylor (Microsoft Knowledge Base Article – 219151, 1999) that demonstrates how to create and manipulate Microsoft Excel files.
- Excel automation using Microsoft J++ (Microsoft Knowledge Base Article – 169796, 2000).

2.7.1 Microsoft Word Automation using C++

This Microsoft Word automation program was developed under an environment that consists of Visual C++ 6.0 and Windows 2000 with MS-Office installed. It describes the implementation of a C++ class that automates Microsoft Word application. The class uses low-level COM and is quite handy in its usage. The C++ class is called CautoWord. It is used to open a word document file and print its contents on the default printer. The class hides all the implementations of low-level COM and provides an easy way for the system developer to invoke the print function provided in the CautoWord class.

The CautoWord class provides the following methods:

- CautoWord() : Constructor function that initializes COM
- int InitAutomation(): initializes the automation. In case of failure, it returns –1 else 0 is returned.
- int PrintDcument(): this function will load the file and print the contents on the default printer. Negative returned value (-1) indicates that an error occurred.

More methods can be added in the CautoWord class for system enhancing purpose. Appendix A contains a usage scenario for Microsoft Word automation program by using the CautoWord class.

2.7.2 Microsoft Excel Automation using Visual Basic

The automation program written in Visual Basic uses Microsoft Excel object library in order to manipulate Microsoft Excel files. A Visual Basic project (standard EXE type) must be created for writing this automation program. In the Visual Basic project environment, a reference call to Microsoft Excel object library must be established. Appendix B shows the steps involved and the source-code for the Microsoft Excel automation program.

2.7.3 Microsoft Excel Automation using Microsoft J++

Basically, the program which is published by Microsoft Corp. shows how to call a COM object (Excel object) from Java. A reference call to Microsoft Excel object library is

made in order to make an Excel application visible and open an existing Excel file. Appendix C shows the steps involved and the source-code for Microsoft Excel automation using Microsoft J++.

However, in January 2001, Microsoft Corp. has announced discontinuity support for MS Java Virtual Machine (MSJVM). This change is the result of a settlement agreement reached in January 2001 that resolved a legal dispute with Sun Microsystems (MSJVM Transition FAQ, November 2003). After September 30 2004, Microsoft Corp. will no longer be authorized to support MSJVM.

Therefore, more efforts and research need to be carried out in order to write an Object-Oriented program that can automate Microsoft Excel documents.

2.8 Summary

This section describes all the aspects that have been studied in order to complete the development of the Object-Oriented Data Automation System for Microsoft Excel Files. With the extensive usage of Microsoft Excel application in business world today, it is crucial to develop a tool that is platform-independent for the ease of data sharing among a group of users.

Microsoft has developed a group of COM objects that gives application developers from worldwide to be able to access its Office applications such as Excel, Words, and Access.

However, these COM objects can only be accessed by using Microsoft COM Wrappers that run under the environment of Microsoft Java Virtual Machine (MSJVM).

Nowadays, a good programming concept "write once, run anywhere" has been widely promoted. Therefore, the Java technology, which provides a standard Java Virtual Machine (JVM), can be used to call the underlying COM objects.