“Active X vs. JavaBeans”

Auteurs Daan de Jong 0433799
Vinesh Kali 0436704

Plaats/Datum Nijmegen, 14 December 2004
Opleiding Informatica, Radboud Universiteit Nijmegen

Versie 14-12-2004

TABLE OF CONTENTS

1. INTRODUCTION

In large application software often two or more applications must be connected and of course this connection must be as efficient as possible. Middleware is the solution to solve such problems. It is most often used to support complex, distributed applications and various platforms. Another reason for using middleware is to centrally provide high-level abstractions and services to applications, to ease programming and integration.

For this study we have investigated the component models Enterprise JavaBeans by Sun Microsystems and ActiveX by Microsoft. Enterprise JavaBeans and Active X are two component models which can be used as middleware. The main goal for this assignment is to gather knowledge of those two middleware techniques, their differences, similarities and possibilities. This assignment is completed by doing a literature study and a case study to experience the two techniques as a software solution.

In chapter two the goals of the assignment is specified and the different research fields are summed up. In chapter three the properties and facts of Enterprise JavaBeans is explained. ActiveX is described in chapter four and the differences and commonalities of the two techniques are summed up in chapter five. The case study is described in chapter six. The conclusions of the theoretical and practical study is described in chapter seven and that is probably the most interesting chapter. To complete this report chapter eight contains the appendices.

2. GOAL

The goal of this research is to investigate how the component models of Enterprise Java Beans (EJB) and ActiveX act as a middleware solution, which one gives the best solution in which situation and what are their commonalities and differences.

To find out what this is all about we need to know what EJB and ActiveX really are. This research will be done by means of a set of research questions:

- How do Enterprise Java Beans compare to ActiveX controls as a middleware solution?
- How does ActiveX refer to .NET

First there shall be a research of: what are EJB and ActiveX, which functionality have they and how can they be used. Finishing this small research we can make a theoretic comparison between these models. After that a simple application is presented, which will work as a middleware solution with two different implementations based on these models. Different situation are reproduced, such as bridging, distribution and stand-alone usage of the models. Finally, a comparison is made of the models using the theoretical an practical information.

Because there are many points which can be researched on EJB and ActiveX the following points decided most important and interesting:

1. Learning curve
2. Development environment
3. Implementation
4. Security
5. Distribution
6. Language independence
7. Platform dependence
8. Bridging

There is not enough time to research and compare all the points in practice. Practical comparison will only be done on the following fields:

- Learning curve
- Development environment
- Implementation
- Deploying
- Bridging

3. ENTERPRISE JAVABEANS

3.1 WHAT ARE ENTERPRISE JAVABEANS?

Enterprise JavaBeans (EJB's) have been introduced in 1998 and represents a standard component architecture for building distributed object-oriented business applications in the Java programming language that can be visualized and interactively manipulated in builder tools. Builders can range from simple lay outing tools to extensive, visual, component-based programming environments.

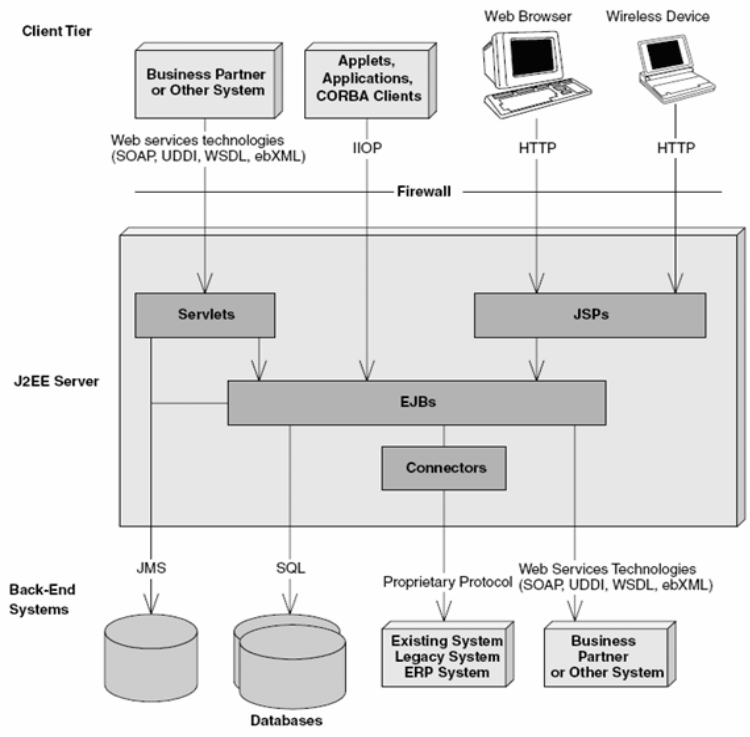

Figure 1: EJB

Enterprise JavaBeans supports the following concepts:

- *properties*:
Properties can be set interactively by the user and be used for the communication among beans. A distinction is made among *standard properties*, *indexed properties* (property arrays), *bound properties* (properties signaling their modification) and *constrained properties* (modifications of these properties can be vetoed by other components).

- *events*: Events are used as a communication vehicle among beans.
JavaBeans is based on Java's object model, which allows the usage of classes and objects in a well-known manner. Interfaces to beans are defined by a set of methods that define properties and events of beans. Interfaces of beans may be deducted from

interfaces of Java classes, which can be done by means of the reflection and introspection mechanism. However, interfaces of beans have to adhere to certain conventions, called patterns in the EJB terminology.

For example, properties are defined by a pair of setter- and getter-methods, a property *name* is defined by two methods *setName* and *getName*, presuming they have the right parameters.
Rather than adhering to naming conventions a EJB can also be defined by an additional BeanInfo class providing meta information about the EJB. A EJB can be represented by a single simple Java class. A bean can also comprise many Java classes as well as resources like images and videos. Complex EJB's are usually stored in and distributed by means of compressed archives, i.e., JAR files. Java's serialization mechanism is used to make bean instantiations persistent. At run-time, Java's garbage collection is used to get rid of any bean instantiations that are not in use anymore.

3.2 ENTERPRISE JAVABEANS AS MIDDLEWARE

The Enterprise JavaBeans were specially designed for distributed systems. The EJB's may communicate across process boundaries, e.g., over the Internet through the standard Java Naming and Directory Interface TM (JNDI).
This is part of the application server which runs on different java *virtual machines* (JVM).

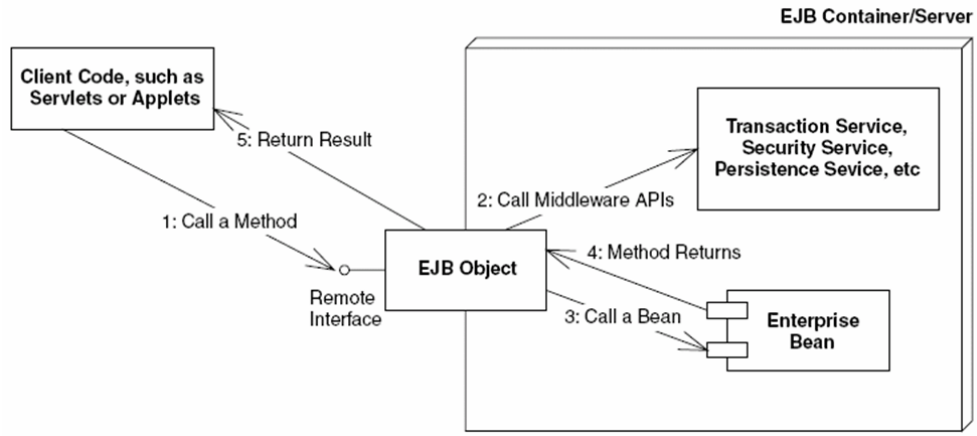

Figure 2: EJB as middleware

3.3 PLATFORMS

Because JavaBeans are just Java classes written to a specific standard, you can use them on any platform for which there is a Java Virtual Machine that supports JDK 1.1–compatible programs. This means that Windows, Macintosh, and most versions of Unix can run Java programs that take advantage of JavaBeans.

3.4 Development of Enterprise JavaBeans and Clients

To create EJB's, Sun provides Software Development Kits (SDK's).
There are two SDK's needed for developing EJB:

1. Java Standard Edition (J2SE) SDK
2. Java Enterprise Edition (J2EE) SDK

The first one is to implement standard Java applications using the standard java model but we also need the Java Enterprise Edition (J2EE) SDK to have access to the EJB packages.
The created EJB has to be distributed on an application server. There are many application servers most of them are quite big. For this research there is make use of JBoss: an open source application server. The application server takes care that the bean is accessible through JNDI for example.

The client can be developed in Java as well, for this only the J2SE SDK is needed. Clients need to lookup the EJB through JNDI so that they can be referenced and their methods are available for the client.

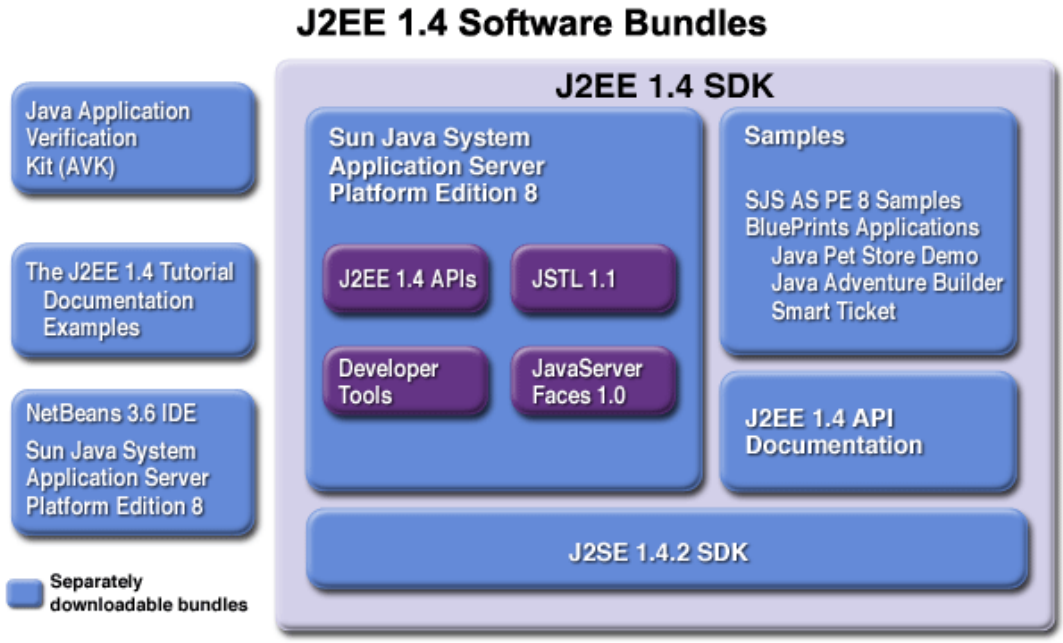

Figure 3: J2EE

4. ACTIVEX

4.1 WHAT IS ACTIVEX?

ActiveX technology is a component-based architecture that enables a developer to use the hundreds of available components. The ActiveX components can interoperate across networks with clients and servers and communicate over the intranet/Internet. ActiveX technology can be used to create, integrate, and reuse software components or "controls" over the Internet or intranets. ActiveX software components, can be integrated into Web pages or any host that supports ActiveX controls.

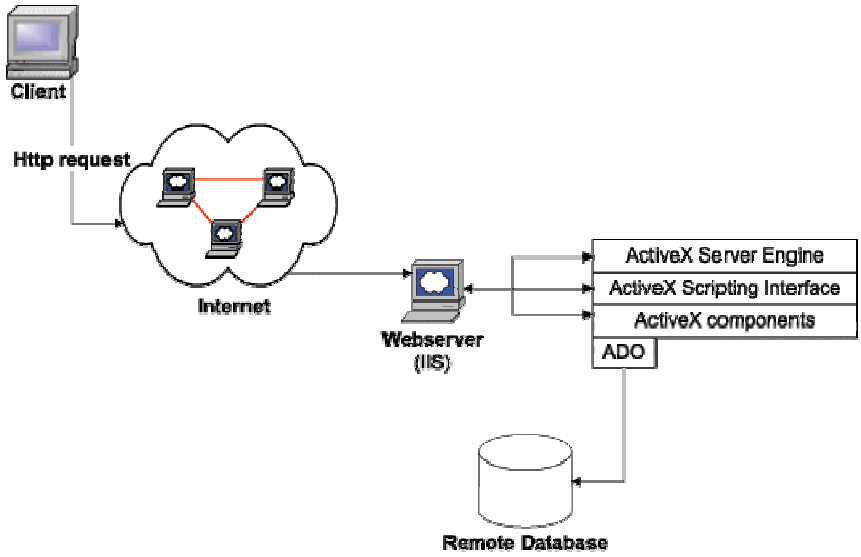

Figure 4: ActiveX

4.2 COMPONENT OBJECT MODEL (COM)

ActiveX controls are based on the component object model (COM). As they are COM objects, the controls can be integrated together with scripting to interact with users at the client side and at the server side. The component object model by Microsoft consists of components, interfaces and a mechanism to find and use components as well as various services. Components represent implementations of various interfaces. A COM interface defines a set of functions one of the most important interfaces is IUnknown. An ActiveX Control is just like COM but implemented an extra interface: IDispatch. This interface makes it possible to invoke methods inside the ActiveX Control.
A unique class identifier (CLSID) is assigned to every component and a unique interface identifier (IID) to every interface. Interfaces are being described by the Microsoft interface description language (MIDL) and are independent of any programming language.

ActiveX objects provided by the ActiveX Control which may run in the same address space, being realized as a dynamic link library (DLL). A ActiveX Control may also run on the same or on a different machine as the client.

In order to get a component with a specific class identifier at run-time, the *procedure Co-CreateInstance is* called. CoCreateInstance requires a class identifier and an interface identifier as parameters. It creates a new instantiation of the specified class and returns an interface of the type required. The client always receives a pointer to the requested interface, but never a pointer to the component itself.

The registry is used in order to find the ActiveX Control with a certain class identifier. The registry determines which server is capable of creating a specific object. When necessary, the specific server is being started. For in process servers, i.e., servers running in the same address space than the client, the appropriate dynamic link library is loaded. The ActiveX Control contains a separate class factory object for each class in order to create instances of that class.

4.3 PLATFORMS

ActiveX components won't work on all platforms because it is developed for Microsoft Windows. Because ActiveX are COM objects it has been ported to other operating systems like Unix en its variants.

4.4 PROGRAMMING LANGUAGE

ActiveX Controls can be programmed in any programming language as long as the generated object code adheres to the binary standard of ActiveX. ActiveX is supported by many languages and tools like Visual C++, Visual J++, Visual Basic, Borland Delphi and PowerBuilder.

4.5 .NET

A few years ago Microsoft introduced a new framework named the .NET Framework. This framework contains many languages which are object oriented and can be used crossly. The framework contains a lot of namespaces (library's) which can be used to program on the standard protocols and make applications for all standard devices.

These new program languages can refer ActiveX Controls but it is not possible to make a new ActiveX Control. Instead of this Microsoft introduced a Class Library.
This is resulted also in a .NET assembly that can be referenced by all .NET supported programming languages. But before an ActiveX Control can be used it must be converted to a .NET assembly. A .NET assembly can be seen as the well known dynamic link library's (DLL). This new assembly contains a interface and a wrapper class for all the methods in the ActiveX Control. If a .NET client creates an instance of this wrapper class it is possible to call the methods from the ActiveX Control.

Besides the .NET Framework Microsoft has developed a .NET Framework SDK and Visual Studio .NET. The SDK provides tools and samples code. One of these tools is AxImp which helps the developer to converted the ActiveX Control into an assembly. In Visual Studio .NET it is possible to create a reference to an ActiveX Control and all the needed assembly's will be generated automatically.

5. ENTERPRISE JAVABEANS AND ACTIVE X COMPARED

5.1 LANGUAGE INDEPENDENCY

An advantage of ActiveX is its programming language independence. Components can be developed in the preferred programming language which allows the developer to choose her preferred programming language and paradigm. The interfaces of a COM component are described with MIDL. The usable data types are restricted to the available programming language – MIDL mapping. Normally JavaBeans are written in Java and the code is compiled to Java byte code. JavaBeans can be written with other languages as long as the compiler compiles the programmed code into Java byte code.

5.2 PLATFORM INDEPENDENCY

One important aspect of the popularity of Java is its operating system independence. This independence is still not reached at all points and a final test of software systems on at least Unix and Windows platforms is required. ActiveX has been built for one platform, i.e., Microsoft Windows. But COM has been ported to UNIX operating systems like Sun Solaris and DEC Alpha.

5.3 BRIDGING ENTERPRISE JAVABEANS AND ACTIVEX

The dominant desktop environment of Microsoft are built to use ActiveX components and not to use EJB. So EJB developers have been quick to provide a solution: the EJB bridge for ActiveX.

This solution enables any EJB to be used as a first-class ActiveX component -- that is, it lets you mix EJB's with ActiveX components. The bridge run-time looks to the system like an ActiveX control, while at the same time giving the Bean a Java environment. The package contains a utility that allows the capabilities of a EJB to be expressed in terms of OLE-type-library and Windows-registry information. After the definition of the Bean is added to the Windows system, programs can create instances of it, and tools such as Visual Basic can interrogate its properties. The bridge run-time code then provides the mappings among the EJB, OLE events, and method calls so that everything is transparent to both the user and the EJB. This is significant: Creators of EJB's don't need to know whether their EJB's are in an ActiveX container. They don't have to write any extra or specialized code to enable their EJB to be used as ActiveX components.

EJB's also provides tools for taking existing ActiveX components and turning them into EJB's in the form of the JavaBeans Migration Assistant for ActiveX. The Migration Assistant analyzes an ActiveX control and provides the skeleton implementation of a Enterprise JavaBean that exhibits the equivalent public interface. It isn't able to convert the implementation code to Java, but it provides a quick start.

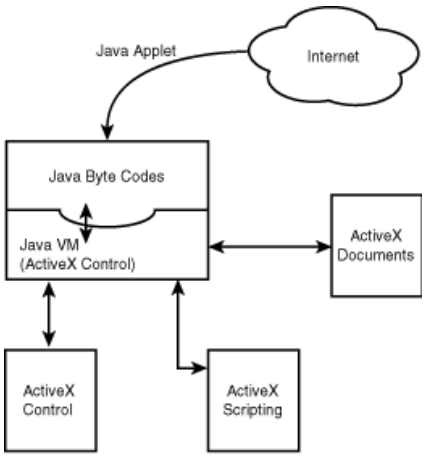

Figure 5: Bridging Enterprise JavaBeans and ActiveX

Microsoft is developing further and further in .NET. At this time Microsoft has a Java Language Conversion Assistant (JLCA) that is able to convert most Java applications written for J2SE to C#. The output can be a executable application or a class library. Soon they will release a new version of JLCA which support J2EE applications like EJB to be converted to C#. So ActiveX may not exists any more over maybe ten years but than .NET is still out there. But also Sun is busy developing with a new version of EJB 3.0 which contain a lot more functionality and a better performance.

5.4 SECURITY

Java is similar to ActiveX in one respect in that it allows downloading and remote execution of code. However, that's where the similarity ends.
Java is both a programming language and an execution environment and is developed with security matters in mind. The Java program runs in a Java Virtual Machine (JVM), which contains the running program within a "sandbox," so it can access only those resources to which it is specifically authorized. The security model of Java offers a mechanism to prevent external or untrusted code to perform system critical operations. ActiveX components are dynamically loaded and are not restricted in their functionality.

6. ENTERPRISE JAVABEANS VS. ACTIVEX: A CASE STUDY

Some characteristics of ActiveX and Enterprise JavaBeans cannot be evaluated by using just the theory. For these a case study is set up where two fictive companies have to solve a slightly simple problem where they evaluate ActiveX, Enterprise JavaBeans and bridging of the these two.

6.1 BACKGROUND

Assume there are two companies, named Company A and Company B. Both company's are using the RSA-encryption method to encrypt their private data before saving this data into their databank.
After a corruption scandal, Company B went bankrupt and Company A decides to take over Company B. After the fusion, the management decides that all software systems must be connected to improve the communication flow in the new raised company named Company C. After analyzing the systems of both companies they found out that the encryption application of Company A is using Enterprise JavaBeans and the application of Company B is using an ActiveX control. These situation is structured as follows:

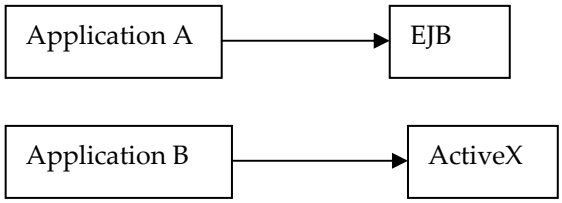

Figure 6: Old situation

The software experts decide there are three possible solutions to this problem:
Bridge the two different systems

The preferred (bridged) situation is as follows:

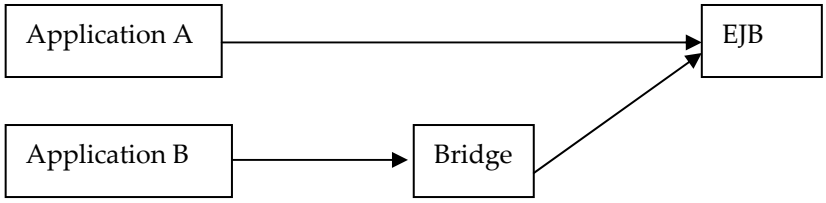

Figure 7: New situation

6.2 APPLICATION

The encryption method RSA uses a one-way function to make it impossible to determine the private key. The one-way function in RSA is multiplication of prime numbers. The function which checks if a given number is prime numbers for the RSA-encryption. The JavaBeans consists of a JAVA-class and the ActiveX .dll of a C++-class with the functionality to check if a number which comes from the Client-application is

a prime number. The Client application(Enterprise JavaBean in JAVA, ActiveX in C#) is the GUI where the user can give a random number as input. The result of the function will also be shown in de Client-application. The sourcecode of the application can be found in de appendix.

6.3 IMPLEMENTATION

6.3.1 Use an Enterprise JavaBeans

This is the standard use of a client connecting to an Enterprise JavaBean

- **Environment:**

The following software is used to develop the JavaBean:
- Java Standard Editon SDK 1.4.206 (J2SE SDK)
- Java Enterprise Edition SDK 1.3 (J2EE SDK)
- Application server: JBoss
- Developer kit: IDE Borland JBuilder

Simple representation of the Enterprise JavaBean setup:

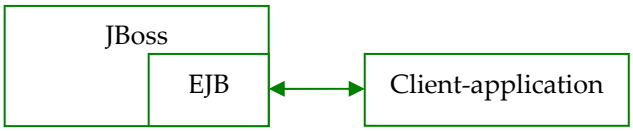

Figure 8: EJB structured

- **Implementation:**

The first step is to configure the SDK's and the application server in JBuilder.
This is not quite easy when your not familiar with JBuilder. The greatest problem was here that JBuilder does not support JBoss this can be solved using a third party software: JBoss Open Tool. If this is installed it is possible to configure JBuilder so that EJB's can be deployed to JBoss.

JBuilder support EJB's very well. It has a special wizard for creating them. There is also a way to create the EJB without this wizard than it is possible to create the EJB in a special EJB designer. This is a graphical environment in JBuilder where after creating the EJB it gives a advantage of adding methods, events and set properties of the EJB.
Both the wizard and the designer generates for each EJB two interfaces and a class.
The first interface called the Home Interface which takes care that it is possible to create a EJB. The second interface is a EJBObject where all the methods from the EJB are declared. The class must contain the implementation of each method. Thanks to the base classes there is no need to put some information about other things like the connection for example.
Beside these source files JBuilder will generate a ejb-jar file which contains all the information over the EJB and their interfaces.
When everything is implemented well the deployment to JBoss can be started.

In this case it just mean to copy the jar-file (which contains all files described above) to the deployment directory of JBoss. After this is finished JBoss will start reading the file and makes it available.

Now it's time to make a client application that will create an instance of the EJB and will give the result of a method after calling it. The client will lookup the EJB on the application server. Then it will reference the home interface and create a instance of the EJB.

When the client was developed there was a little problem: the client was unable to lookup the EJB on the application server.
The cause of this problem was that in the designer there was created a local EJB. But local is does not mean on the local PC or in the local LAN but on the local application server. Assuming that for example a JSP client is also hosted on the same server as the EJB. To reference the EJB from a standalone client it is needed to create an remote EJB!

- **Developer environment:**

Developing an Enterprise JavaBean can be quite easy but therefore it is needed to have a lot of knowledge of the IDE in this case JBuilder. Without this it might be easier to create the java source file with a simple text editor. Borland has put a lot of useful things in their environment but it might be a good advise to read the manual first before starting otherwise it is possible to get confused.

- **Learning curve:**

It is possible to create an EJB just with only knowing the basics of development in Java. Like using interfaces, classes and use the advantages of the packages from the SDK's. This all makes it relative easy to create an EJB without an special IDE.
This is a very strong point which give the developer not the idea that he/she has to study a lot before the developer can actually start with the implementation.
A weak point is that besides java the developer has to know the used application server as well.
It sounds maybe a little bit strange but developing a client application is much more difficult then developing a EJB.

6.3.2 Use an ActiveX control

This is the standard use of a client connecting to an ActiveX Control

- **Environment:**

The following software is used to develop the ActiveX:
- Developer kit for ActiveX: Visual Studio
- .NET Framework 1.1
- .NET Framework SDK
- Developer kit for Client application: Visual Studio .NET

Visual Studio is used for developing the ActiveX control because it is the most used software to develop ActiveX controls by companies so it represents a good practical environment. The program language Sharp together with Sharp developer is used to

develop the client application. Sharp is a relative new program language based on .NET. CSharp is chosen because the project duration was short (ca. 6 weeks) and the learning curve of CSharp is also very short.

Simple representation of the ActiveX setup:

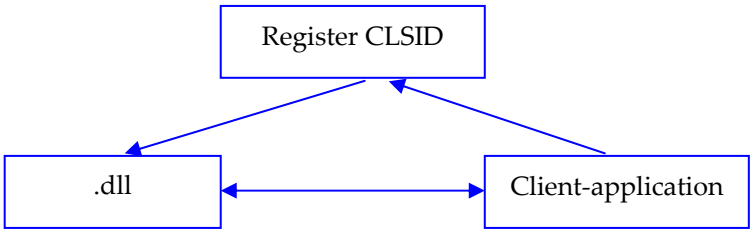

Figure 9: ActiveX structured

- **Implementation:**

Visual Studio has a complete ActiveX wizard implemented. By walking the wizard all necessary ActiveX code will be generated dependent on the input of the user. This input is only highly necessary configuration of the ActiveX control like controlname and server type.
After the ActiveX control is generated a lot of files are automatically created. The generated code exists of two type libraries, three packages and four global files. The three packages all have a different functionality namely 'application' where the main controls of the ActiveX are place like the registration of the ActiveX control, 'control' where the actual functionality is nested and 'property' to configure your ActiveX control. The source code of the application must be placed in the 'control' package and after this the ActiveX is finished.
Visual Studio contains a very smart tool to test your ActiveX control without writing a client application. This module is called the 'ActiveX Control Test Container' The first step is to load the ActiveX control into the container and registrate it. After that the container generates a client application due to the input parameters of the function which is inside the ActiveX control. After entering the input parameters the container uses the registered ActiveX control and returns a return value.

- **Developer environment:**

Creating an ActiveX control using Visual Studio is much simpler then thought, but a few points are rather hard to understand. The content and function of the generated files is hard to understand as a result of the missing of sense full comment lines. It is just to must work to understand all of the created code, because of its difficulty and therefore It takes a long time to find out where the functions of the application must be placed in the generated code.

6.3.3 ActiveX in .NET

Structured representation of ActiveX in .NET.

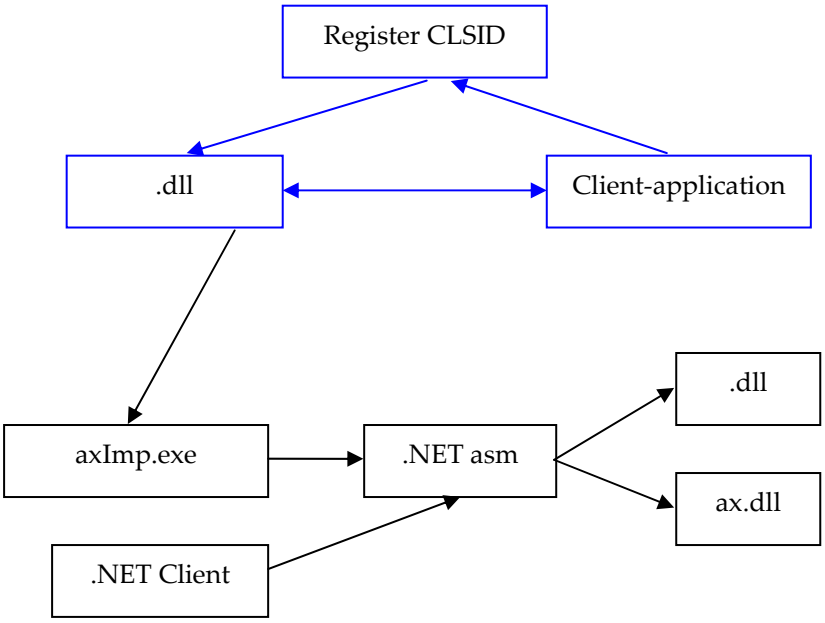

Figure 10: ActiveX in .NET

De .dll which includes the functionality is being transformed to two '.NET dll's' in two steps. The executable axImp.exe uses the .dll to generated a standard .NET assembly file. Any .NET client application can now use the assembly file to use the function. Two .dll are generated by the assembly file to let other applications use the function which aren't designed in .NET.

A client application must be written to actually use the ActiveX control. This client application is written in .NET using 'Csharp' to test whether the ActiveX control works well in .Net. Before it is possible to use the ActiveX Control it need to be reference the created assembly's. When a ActiveX Control is referenced an instance can be created. In Visual Studio it is also possible to add an ActiveX Control as a form control. So it is possible to put the control on a form. When the control is placed on the form all assembly's and references are created automatically.

6.3.4 Bridging ActiveX and Enterprise JavaBeans

In the Java SDK's there are also some tools included. One of these tools is java2activex bridge. Through this bridge it is possible to connect to the Enterprise JavaBean with a Client which can only make us of ActiveX and no Java. Or in other words it will create an ActiveX interface/wrapper for the Enterprise JavaBean.

For this bridge the following tools are needed:
- packager.exe (included in the J2SE SDK)

- Some C++ tools and compilers
- the Enterprise Java Bean jar package

The result of the bridge can be sketched as follow:

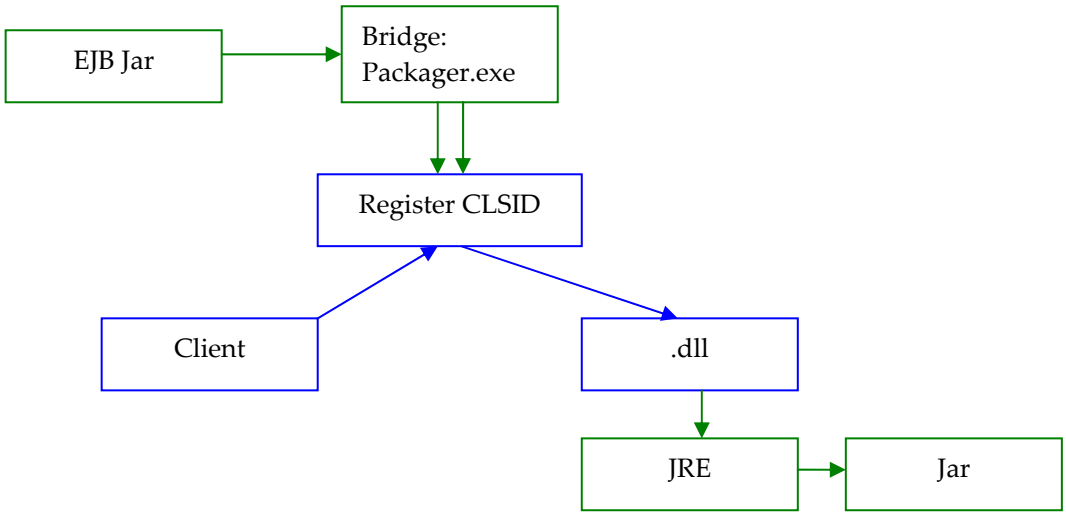

Figure 11: Bridging ActiveX and EJB

To create a successful ActiveX interface there is need to set the right parameters of packager.exe:
Clsid: Create a valid clsid with guidgen.exe.
Out: The output directory must be %JREDIR%\axbridge\bin.
Reg: Register the created ActiveX control in the Windows register.

The bridge needed also some C++ tools and compilers. The most important ones are: Midl.exe: Microsoft IDL compiler, CL.exe: C++ Compiler and link.exe: a linker. Besides these executables there is need for some IDL-files. Packager creates an IDL-file from the EJB class file and together with the other IDL-files the ActiveX control will be build.

The build ActiveX Control can now be tested in the ActiveX Control Test Container form Visual Studio. If everything is alright then the generated ActiveX Control seems to work fine. Although there might be some little trouble with types like a char type in java will be converted as short in C++.

Table 1: ActiveX testcontainer

JAVA	Type Library IDL	Visual C++
Boolean	VARIANT_BOOL	BOOL
Byte	short	Short
Char	short	Short
Double	double	Double
Float	single	Float
Integer (object)	long	Long
Int	long	Long
long (8 bytes)	long (4 bytes)	long (4 bytes)
Object (or subclass)	IDispatch*	IDispatch*
Short	short	Short
String	BSTR	CString (MFC)

6.3.5 Deployment of ActiveX and Enterprise JavaBeans

Like said before EJB are specially designed to be distributed so deploying them is easy to. Placing an application server in the LAN is enough. A client has to know where the application server is running and on which port it is available. In the Microsoft Windows LAN there will always be one or more fileservers or something like that. It is possible to place the ActiveX Control on one of them and all clients in the LAN can make use of it as long as the ActiveX Control is registered in the clients register.
It must be theoretically possible to do this over the internet as well but this is not tested. In the LAN everything works the same as it used before. The only difference between EJB and ActiveX is that the EJB Client must be changed and recompiled and for ActiveX the control has to be reregistered again.
However the bridged EJB must be in a special directory on the same PC as the client. This is because the Java Runtime Engine is installed locally and not remote available. The ActiveX Control can be registered but the JRE will not start so the instance of the ActiveX control cannot be created.

6.4 USING ACTIVEX CONTROLS IF THEY WERE ENTERPRISE JAVABEANS

Although using ActiveX controls is possible with some Java compilers this probably isn't a good idea. One of Java's strengths is its ability to work across platforms. EJB's can be used regardless of platform; however, using an ActiveX control in a Java project (wrapped as a EJB or not) limits your application to environments that support ActiveX—namely Windows.

7. CONCLUSIONS

Both Enterprise JavaBeans and ActiveX are fine middleware products. These two techniques are perfectly developed by Sun and Microsoft to be used in an application environment.

To support many programming languages you better should choose ActiveX but for supporting most platforms a Enterprise JavaBean is a better choice. If the security of the system is a big issue, the Enterprise JavaBean is in a big advance compared to the ActiveX components. A much more interesting point is the difficulty to learn how to use these middleware products. If we take a look only the EJB it is not very difficult if the developer already knows Java. Creating a ActiveX Control is much more difficult because there is much more code to understand. Only be familiar with C++ for e.g. is not enough.

However the client applications are just the opposite. Writing a EJB client is more difficult than writing a ActiveX client. This is just true looking at these two in a basic way without the advantages of development environment. Nowadays the Development Environment in combination with third party tools are very advanced and can generate lots of code. Most of them can create a standard skeleton of source code where the developer just need to write his/her functionality and does not need to have much knowledge of the used techniques like we have seen in Visual Studio and JBuilder.

The new program languages that are gathered in the .NET Framework can refer ActiveX Controls but it is not possible to make a new ActiveX Control. Instead of this Microsoft introduced a Class Library. This is resulted in a .NET assembly that can be referenced by all .NET supported programming languages. In the future the Class Library will be a replacement for the ActiveX Control.

8. APPENDICES

8.1 REFERENCES

Literature

Joe Chavez (2003), "Enterprise JavaBeans", powerpoint presentation

D. Birngruber, W. Kurschl, J. Sametinger, "Comparison of JavaBeans and ActiveX" , paper.

Eric J. Grossman PH.D, "ActiveX Security Issues", paper.

Websites

Sun Developer Network, JavaBeans API,
http://java.sun.com/beans/docs/spec.html

Microsoft, COM: Component Object Model Technologies
http://www.microsoft.com/com/default.mspx

8.2 SOURCECODE

```
// Primenumber function for C++

BOOL CPrimecalculatorCtrl::calcprimes(long number)
{
        boolean flag;
        int d;
        int k;

        if (number == 2)
                flag = true;
        else if ((number % 2) == 0)
                flag = false;
        else
        {
                flag = true;
                k = (int) sqrt((double) number);
                for (d = 3; d <= k; d = d + 2)
                {
                        if ((number % d) == 0)
                                flag = false;
                }
        }
        if (flag)
                return true;
        else
                return false;
}


// Primenumber function for Java

  public boolean calcprime(long number) {
    boolean flag;
    int d;
    int k;

        if (number == 2)
                flag = true;
        else if ((number % 2) == 0)
                flag = false;
        else
        {
                flag = true;
                k = (int) Math.sqrt((double) number);
```

```
            for (d = 3; d <= k; d = d + 2)
            {
                if ((number % d) == 0)
                    flag = false;
            }
        }
        if (flag)
           return true;
        else
          return false;
  }
}
```