Comparisons

All three technologies, RMI, CORBA and EJB's are distributed technologies, in that they can distribute an application across two or more computers.

RMI is the Java facility for invoking methods on objects that live in a different VM. It is a good choice for small network-enabled programs in the client-server model.

CORBA is a language-independent specification for network computing. It enumerates several services: naming, security, persistent objects and so on.

CORBA would be useful in building more complex apps than RMI would easily support alone. Applications which require security authorities and multiple servers.

Enterprise Java Beans are standards for making network-accessible components. They are part of the Java 2 Enterprise Edition, which specifies many of the kinds of services that CORBA specifies and uses RMI to communicate.

EJB's and their related technologies are, like CORBA, useful for large-scale applications. Application scalability refers to the capacity of an app to handle growth, especially in handling more users and evolving concurrently with our business needs.

It is hard to say exactly where one would choose one over the other. Those decisions should be driven by the requirements of the application one is designing.

Examples:

1- Hello world example in CORBA

Defining the Interface (`Hello.idl`)

```
module HelloApp
{
    interface Hello
    {
        string sayHello();
    };
};
```

Implementing the Server (`HelloServer.java`)

The example server consists of two classes, the servant and the server. The servant, `HelloServant`, is the implementation of the `Hello` IDL interface; The servant is a subclass of `_HelloImplBase`, which is generated by the `idlj` compiler from the example IDL. the extra code to deal with the ORB, with marshaling arguments and results, and so on, is provided by the server.

The server class has the server's `main()` method, which:

- Creates an ORB instance
- Creates a servant instance (the implementation of one CORBA `Hello` object) and tells the ORB about it
- Gets a CORBA object reference for a naming context in which to register the new CORBA object
- Registers the new object in the naming context under the name "Hello"
- Waits for invocations of the new object

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

class HelloServant extends _HelloImplBase
{
    public String sayHello()
    {
     return "\nHello world !!\n";
    }
}


public class HelloServer {

    public static void main(String args[])
    {
     try{
         // create and initialize the ORB
         ORB orb = ORB.init(args, null);

         // create servant and register it with the ORB
         HelloServant helloRef = new HelloServant();
         orb.connect(helloRef);

         // get the root naming context
         org.omg.CORBA.Object objRef =
             orb.resolve_initial_references("NameService");
         NamingContext ncRef = NamingContextHelper.narrow(objRef);

         // bind the Object Reference in Naming
         //make sure there are no spaces between ""
         NameComponent nc = new NameComponent("Hello", "");
         NameComponent path[] = {nc};
         ncRef.rebind(path, helloRef);
```

```
            // wait for invocations from clients
                java.lang.Object sync = new java.lang.Object();
                synchronized (sync) {
                    sync.wait();
                }

        } catch (Exception e) {
            System.err.println("ERROR: " + e);
            e.printStackTrace(System.out);
        }
        }
}
```

Implementing the Client Application (`HelloClient.java`)

The example application client that follows:

- Creates an ORB
- Obtains a reference to the naming context
- Looks up "Hello" in the naming context and receives a reference to that CORBA object
- Invokes the object's `sayHello()` operation and prints the result

```
import HelloApp.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class HelloClient
{
    public static void main(String args[])
    {
     try{
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);

                // get the root naming context
                org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService");
                NamingContext ncRef = NamingContextHelper.narrow(objRef);

                // resolve the Object Reference in Naming
                // make sure there are no spaces between ""
                NameComponent nc = new NameComponent("Hello", "");
                NameComponent path[] = {nc};
                Hello helloRef = HelloHelper.narrow(ncRef.resolve(path));

            // call the Hello server object and print results
            String hello = helloRef.sayHello();
            System.out.println(hello);

        } catch (Exception e) {
            System.out.println("ERROR : " + e) ;
            e.printStackTrace(System.out);
        }
        }
```

}

2- Hello world example using EJB

We need three different files:

a. SessionBeanRemote.java:- This is the Remote interface which extends javax.ejb.EJBObject package. These are similar to RMI Remote interface. The use of remote interface is particularly helpful in providing business-specific functionality of an EJB. Here @Remote is the annotation used to declare the interface as Remote. String getAddress(); String getCompanyname(); String getResult():-These are the methods defined in the bean.

```
package ejb;

import javax.ejb.Remote;


@Remote
public interface SessionBeanRemote {

 String getResult();

 String getAddress();

 String getCompanyname();
}
```

b. SessionBeanBean.java:-This is the bean of type session in which we have defined the body of the method which were declared in the interface named SessionBeanRemote.java. @Stateless is the annotation used to declare the bean as a session type.

```
package ejb;
import javax.ejb.Stateless;

@Stateless
public class SessionBeanBean implements
   SessionBeanRemote,SessionBeanLocal {
```

```
 public String getResult() {
 return "Hello World";
 }
 public String getAddress() {
 return "This is my address";
 }
 public String getCompanyname() {
 return "This is my company.Ltd.";
 }
}
```

c. Main. java:-This is the client application from which we can access the methods which are defined in the bean. @EJB is the annotation used for configuring the EJB values for a field and method.

```
import ejb.SessionBeanRemote;
import javax.ejb.EJB;

public class Main {

  @EJB
  private static SessionBeanRemote sessionBeanBean;

  public static void main(String[] args) {
  System.out.println("Displaying Message using EJB:");
  System.out.println("================================");
  System.err.println("Name of the Company is : ="
+ sessionBeanBean.getCompanyname());
  System.err.println("Address of the Company is : ="
+ sessionBeanBean.getAddress());
  System.err.println("Message is : ="
    + sessionBeanBean.getResult());
  System.out.println("================================");
  }
}
```