Core Course XXI--Distributed Technologies

Unit I

Introduction to Distributed Computing – Challenges involved in establishing remote connection - Strategies involved in remote computation – Current Distributed computing practices through Dot Net and Java technologies.

Unit II

Advanced ADO.NET – Disconnected Data Access – GridView, DetailsView, FormView Controls – Crystal Reports – Role of ADO.NET in Distributed Applications.

Unit III

Advanced ASP.NET – AdRotator, Multiview, Wizard and Image Map Controls – Master Pages – Site Navigation – Web Parts – Uses of these controls and features in Website development.

IV

Advanced Features of ASP.NET – Security in ASP.NET – State Management in ASP.NET – Mobile Application development in ASP.NET – Critical usage of these features in Website development.

Unit V

Web Services – Role of Web Services in Distributed Computing – WSDL, UDDI, SOAP Concepts involved in Web Services – Connecting a Web Service to a Data Base – Accessing a Web Service through an ASP.Net Application.

Text Book(s)

1. ASP .NET 3.5, Walther, SAMS Publication, 2005

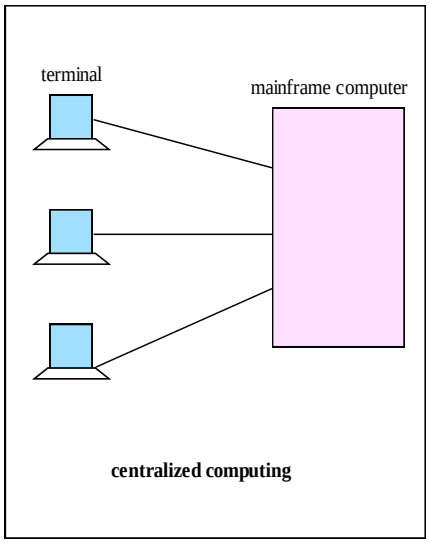
UNIT-I Introduction to Distributed Computing

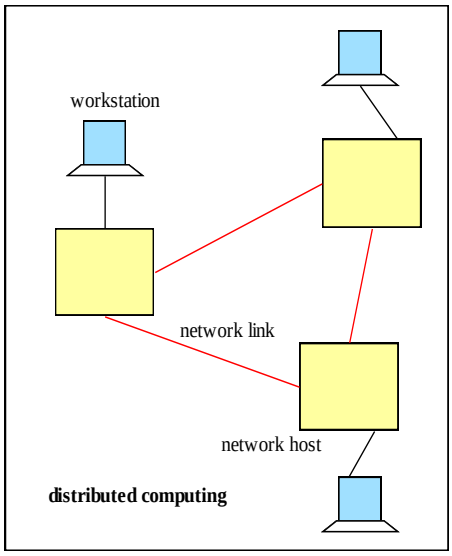
Distributed computing is a field of computer science that studies distributed systems. A distributed system consists of multiple autonomous computers that communicate through a computer network.

A computer program that runs in a distributed system is called a distributed program, and distributed programming is the process of writing such programs

In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.
The processors in a typical distributed system run concurrently in parallel.

A distributed system is a collection of independent computers, interconnected
via a network, capable of collaborating on a task.





Example Distributed systems:

- Internet
- ATM (bank) machines
- Intranets

Goals/Benefits:

- ✓ Resource sharing
- ✓ Scalability
- ✓ Fault tolerance and availability

- ✓ Performance
- ✓ Economics
- ✓ Speed
- ✓ Inherent distribution
- ✓ Reliability
- ✓ Incremental growth

Disadvantages:

- ✓ Software
- ✓ Network
- ✓ More components to fail
- ✓ Security

- ❖ Parallel computing can be considered a subset of distributed computing

Challenges involved in establishing remote connection

The challenges involved in remote connection are as mentioned below.

• Heterogeneity
• Openness
• Security
• Scalability
• Failure handling
• Concurrency
• Transparency

Heterogeneity

• Different networks, hardware, operating systems, programming languages, developers.
• We set up protocols to solve these heterogeneities.
• Middleware: a software layer that provides a programming abstraction as well as masking the heterogeneity.
• Mobile code: code that can be sent from one computer to another and run at the destination.

Openness

• Make it easier to build and change
• The openness of DS is determined primarily by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.
• Open systems are characterized by the fact that their key interfaces are published.
• Open DS are based on the provision of a uniform communication mechanism and published

interfaces for access to shared resources.

• Open DS can be constructed from heterogeneous hardware and software.

Security

• Security for information resources has three components:

- Confidentiality: protection against disclosure to unauthorized individuals.

- Integrity: protection against alteration or corruption.

- Availability: protection against interference with the means to access the resources.

• Two new security challenges:

- Denial of service attacks (DoS).

- Security of mobile code.

Scalability

• A system is described as scalable if it remains effective when there is a significant increase in the number of resources and the number of users.

• Distributed system should be more reliable than single system.

• Challenges:

- Controlling the cost of resources or money.

- Controlling the performance loss.

- Preventing software resources from running out

- Avoiding performance bottlenecks.

Failure handling

• When faults occur in hardware or software, programs may produce incorrect results or they may stop before they have completed the intended computation.

• Techniques for dealing with failures:

- Detecting failures

- Masking failures

- Tolerating failures

- Recovering from failures

- Redundancy

Concurrency

• There is a possibility that several clients will attempt to access a shared resource at the same time.

• Any object that represents a shared resource in a distributed system must be responsible for ensuring that operates correctly in a concurrent environment.Redundancy improves it.

Transparency

• Transparency is defined as the concealment from the user and the application programmer of the separation of components in a distributed system, so that the system is perceived as a whole rather than as a collection of independent components.

• Eight forms of transparency:

- Access transparency
- Location transparency
- Concurrency transparency
- Replication transparency
- Failure transparency
- Mobility transparency
- Performance transparency
- Scaling transparency

Strategies Involved In Remote Computation

The Remote Computation System (RCS)

- Today many high performance computers are reachable over some network.
- However, the access and use of these computers is often complicated.
- This prevents many users to work on such machines.
- The Goal of the Remote Computation System (CS) is to provide easy access to modern parallel algorithms on supercomputers for the inexperienced user.

- Wide area computer networks have become a basic part of today's computing infrastructure.
- These networks connect a variety of machines, from workstations to supercomputers, presenting an enormous computing resource.
- Furthermore, sufficient software for solving problems in numerical linear algebra on high performance computers is around today.
- However, the access and the use of these computers and the software is often complicated.
- A major problem for the inexperienced user to exploit such high performance computers is that he has to deal with machine dependent low level details.
- The goal of this project is to make high performance computing accessible to scientists and engineers without the need for extensive training in parallel computing and allowing them to use resources best suited for a particular phase of the computation.
- Also, the emphasis is laid on algorithms for solving problems in numerical linear algebra, the concepts presented here are applicable to any high performance algorithms
- This goal shall be achieved with a remote computation system (RCS), which provides an easy-to-use mechanism for using computational resources remotely.
- The user's view of the RCS is that of an ordinary software library.
- The user calls RCS library routines (e.g. to solve a system of linear equations) within his program running on a workstation.
- In contrast to common libraries, the problem is not necessarily solved on the local workstation, but is dynamically allocated on an arbitrary machine in a given pool of computers, in order to minimize the response time.
- Because RCS is called asynchronously, it allows distributed applications with several solvers running concurrently on different computer platforms.
- The Remote Computation System consists of two components
    - A library of interface routines
    - The run time system.

- The underlying computational software can be any existing scientific package such as LAPACK
- Before running a RCS application, the user first has to start up the RCS run time system.
- RCS is a single user system but multiple RCS applications are allowed per user to run concurrently.
- The server is the core of the RCS. Its task is to accept requests from user's application and to start an appropriate solver on a host in the pool.
- If the remote host is not specified by the user, the server selects the solver-host pair such that the response time is minimized
- Such a selection process has not yet been done in the context of a numerical library.
- In order to make of an optimal choice, the server needs information about
    - The problems which RCS can solve
    - The host computers in the pool and their characteristics such as number of processors etc.
    - The available computational software (solvers) on each host and their characteristics. For instance, a theoretical model is required to assess its response time
    - Dynamic parameters as the current workload on each host and the available communication bandwidth on the network.
- A daemon called monitor on each host is responsible for periodically measuring the dynamic parameters.
- All other information is static and is read from a configuration file at startup time.
- With Remote Desktop, you can have access to a Windows session that is running on your computer when you are at another computer.
- This means, for example, that you can connect to your work computer from home and have access to all of your programs, files, and network resources as though you were sitting at your computer at work.
- You can leave programs running at work and when you get home, you can see your work desktop displayed on your home computer, with the same programs running.

- You can keep your programs running and preserve the state of your Windows session while another user is logged on. When that user logs off, you can reconnect to your session in progress.
- To use Remote Desktop, you need:
- A computer ("host" computer) running Windows XP Professional with Service Pack 2 or Windows Server 2003 with Service Pack 2 ("remote" computer) with a connection to a local area network (LAN) or the Internet.
- A second computer ("client" computer) with access to the LAN via a network connection, modem, or virtual private network (VPN) connection. This computer must have Remote Desktop Connection installed.
- Appropriate user accounts and permissions.

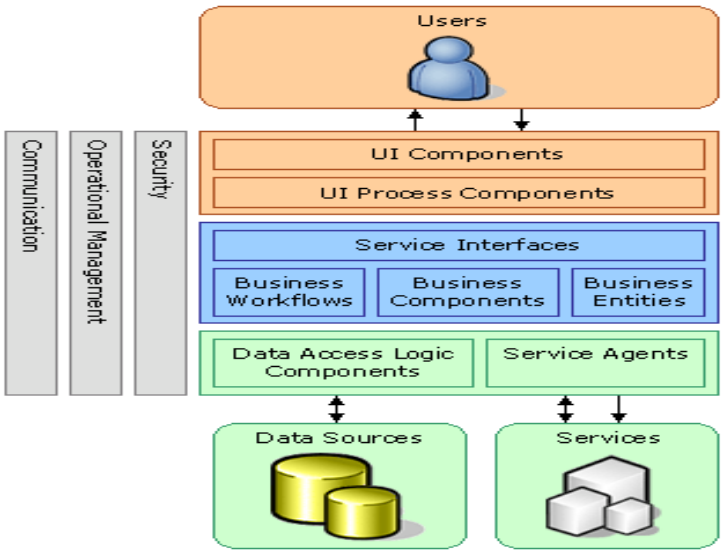
Distributed Computing Using .NET Remoting

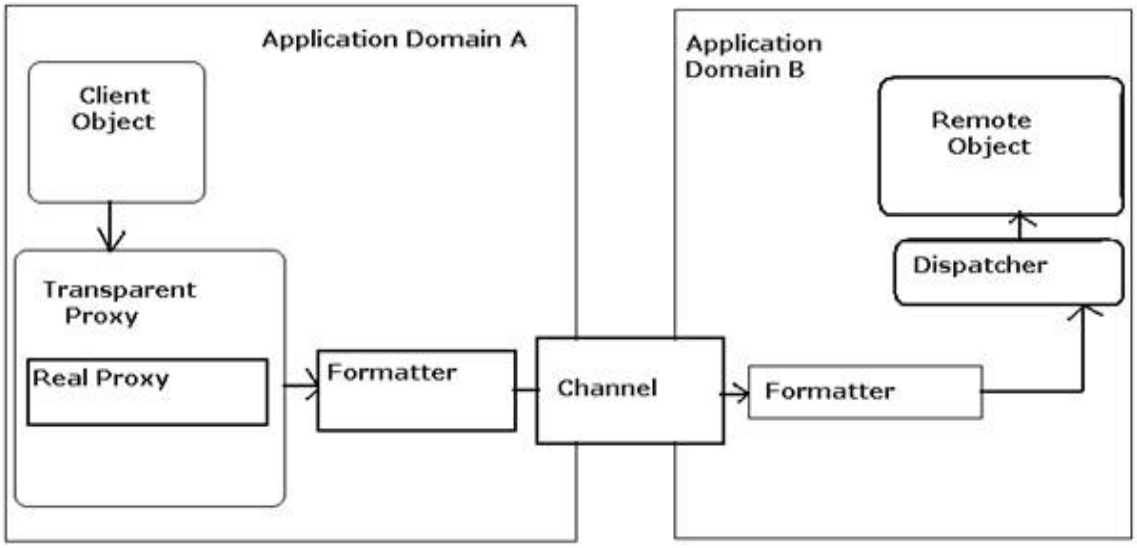
Distributed computing has become the identity of present generation software applications. In past, developers used technologies like DCOM (Distributed Component Object Model) by Microsoft, CORBA (Common Object Request Broker Architecture) by OMG and Java RMI (Remote Method Invocation) by SUN for the same purpose.

Microsoft .Net Remoting is an extensible framework provided by Microsoft .Net Framework, which enables communication across Application Domains (AppDomain).

AppDomain is an isolated environment for executing Managed code. Objects within same AppDomain are considered as local whereas object in a different AppDomain is called Remote object. Microsoft .Net Remoting comes into picture when an application requires communication between different AppDomains.

REMOTING IN .NET





Just like other distribute computing technologies, in .Net Remoting also, client object doesn't make a direct call to the remote object, rather it creates a proxy object of the remoting object and then uses the proxy object to invoke methods of remote object. When the client object calls a method of remote object via proxy, the call is formatted by a formatting object (SOAP, Binary or any Custom formatter). After formatting the call is transferred to the remote object via proper channel (TCP Channel, HTTP Channel or any Custom channel) where the method is executed. Now the entire process is reversed to return appropriate result to the client object.

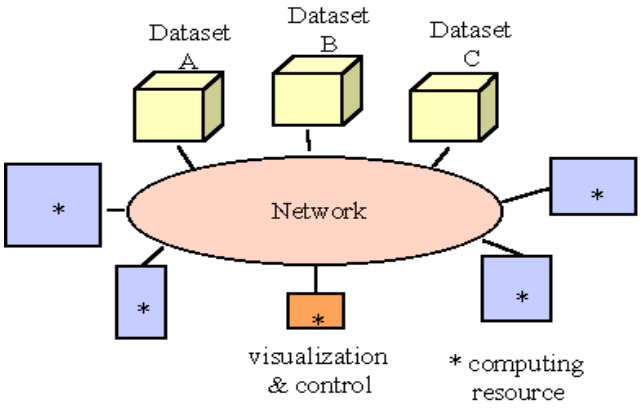


Figure 1: .Net remoting Architecture

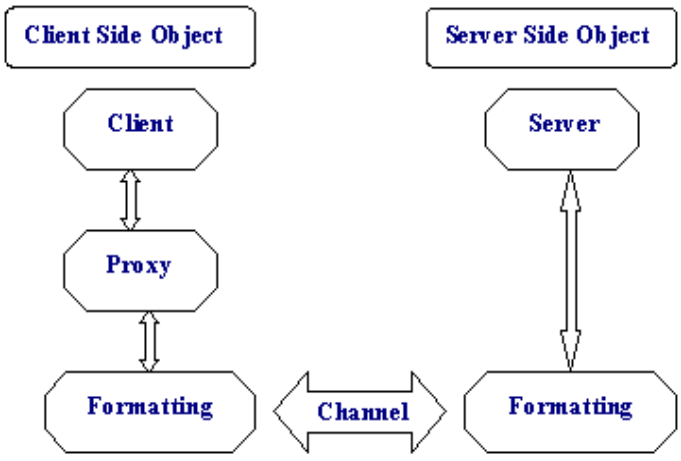
Remote Object

Remote object (Located at server side) is derived from System.MarshalByRefObject class which provides required functionality for communicating with an object in different AppDomain. Any object that needs to be transferred across appDomains has to be passed by value and should implement ISerializable interface. An object which doesn't implement ISerializable interface can't be transmitted across appDomains.

Example:

```
using System;
namespace Employee
{
public class Employee: MarshalByRefObject
{
//Constructor
public Employee
{
// Implementation details
}
//Other functions will come here
}
}
```

Proxy Object



This is created when a client object activates a remote object.

We can have two types of remoting objects i.e. Client Activated Object or Server Activated Object.

Client Activated Object: Client Activated Remote object is one whose life is controlled by client object. Here one single instance of remote object will exist per client object. Client Activated Object is created using new keyword. Client Activated object can store state information for a specific client.

Server Activated Object: Contrary to Client Activated Remote objects, life time for Server Activated Object is controlled by Server. These objects are created when client object calls a method on the proxy object. There are two types of Server Activated Objects i.e. SingleCall and Singleton.

SingleCall: They serve only one client request. Once the client request is over, they are subjected to garbage collection. They don't store any state information.

Singleton: They serve multiple clients, thereby allowing information sharing between requests. They are stateful objects unlike SingleCall, which is stateless.

```
//Creating a new instance of a remote object using new.
Employee Emp = new Employee();
//Creating a new instance of a remote object using CreateInstance.
Employee Emp = (Employee)Activator.CreateInstance(...);
```

Note: Above method creates an instance of the remote object based on the parameter passed. For a complete listing of the same, please refer MSDN.

```
//Retrieving an Existing Instance.
Employee Emp = (Employee)Activator.GetObject( typeof(Employee), HTTP://[Path]);
```

All the calls to the remote object (at server side) are routed through proxy object. To make things little complicated, there are two types of proxy objects involved in the process i.e. Transparent proxy and Real proxy. Transparent proxy provides the implementation of all public method to Client object which means that client object always talks to Transparent proxy which in turn makes call to Real proxy. Real proxy passes the message to channel object. Developers can customize the Real proxy to include additional functionalities if required.

Formatters

They encode and decode the message between client application and Remote object. .Net Framework provides SOAP and Binary formatter. It also supports custom formatters (IRemotingFormatter) developed by programmers.

Binary Formatter*:* System.Runtime.Serialization.Formatters.Binary
SOAP Formatter*:* System.Runtime.Serialization.Formatters.Soap

Channels

They are responsible for transmitting the message over the network. .Net Framework provides HTTPChannel and TCPChannel. It also supports custom channels (IChannel) developed by programmers.

HTTPChannel: System.Runtime.remoting.Channels.Http
TCPChannel: System.Runtime.remoting.Channels.Tcp

By default HTTP Channel uses SOAP Formatter and TCP Channel uses Binary Formatter.

Channels need to be registered with the remoting service as shown below.

```
//Registering a channel
ChannelServices.RegisterChannel();
```

Hosting a Remoting Application

Remoting Host is a runtime environment for the remote object i.e. Microsoft IIS Server.

Step By Step: Let's see the entire step once again.

1. Client object registers a channel.
2. Creation of Proxy object (Client activated or Server Activated)
3. Calling the method of remote object via proxy.
4. Client side formatter formats the message and transmits via appropriate channel.
5. Server side formatter reformats the message.
6. The specified function on remote object is executed and the result is returned.
7. Above process of formatting and reformatting is reversed and the result is returned to client object.

Above article explains the basic terms and technology involved in .Net Remoting. It's possible to create complex distributed applications using .Net Remoting. Developers can create their own custom channels and formatters depending on business needs. There is no built in security provided by .Net Remoting framework. The security features need to be provided by the hosting environment.

Distributed Systems in Java:

Java Remote Method Invocation (RMI) allows you to write distributed objects using Java.

RMI provides a simple and direct model for distributed computation with Java objects. These objects can be new Java objects, or can be simple Java wrappers around an existing API. Java embraces the "Write Once, Run Anywhere model. RMI extends the Java model to be run everywhere."

Because RMI is centered on Java, it brings the power of Java safety and portability to distributed computing. You can move behavior, such as agents and business logic, to the part of your network where it makes the most sense. When you expand your use of Java in your systems, RMI allows you to take all the advantages with you.

RMI connects to existing and legacy systems using the standard Java native method interface JNI. RMI can also connect to existing relational database using the standard JDBC package. The RMI/JNI and RMI/JDBC combinations let you use RMI to communicate today with existing servers in non-Java languages, and to expand your use of Java to those servers when it makes sense for you to do so. RMI lets you take full advantage of Java when you do expand your use.

Advantages

At the most basic level, RMI is Java's remote procedure call (RPC) mechanism. RMI has several advantages over traditional RPC systems because it is part of Java's object oriented approach. Traditional RPC systems are language-neutral, and therefore are essentially least-common-denominator systems-they cannot provide functionality that is not available on all possible target platforms.

RMI is focused on Java, with connectivity to existing systems using native methods. This means RMI can take a natural, direct, and fully-powered approach to provide you with a distributed computing technology that lets you add Java functionality throughout your system in an incremental, yet seamless way.

The primary advantages of RMI are:

- **Object Oriented:** RMI can pass full objects as arguments and return values, not just predefined data types. This means that you can pass complex types, such as a standard Java hashtable object, as a single argument. In existing RPC systems you would have to have the client decompose such an object into primitive data types, ship those data types, and the recreate a hashtable on the server. RMI lets you ship objects directly across the wire with no extra client code.
- **Mobile Behavior:** RMI can move behavior (class implementations) from client to server and server to client. For example, you can define an interface for examining employee expense reports to see whether they conform to current company policy. When an expense report is created, an object that implements that interface can be fetched by the client from the server. When the policies change, the server will start returning a different implementation of that interface that uses the new policies. The constraints will therefore be checked on the client side-providing faster feedback to the user and less load on the server-without installing any new software on user's system. This gives you maximal flexibility, since changing policies requires you to write only one new Java class and install it once on the server host.
- **Design Patterns:** Passing objects lets you use the full power of object oriented technology in distributed computing, such as two- and three-tier systems. When you can pass behavior, you can use object oriented design patterns in your solutions. All object oriented design patterns rely upon different behaviors for their power; without passing complete objects-both implementations and type-the benefits provided by the design patterns movement are lost.
- **Safe and Secure:** RMI uses built-in Java security mechanisms that allow your system to be safe when users downloading implementations. RMI uses the security manager defined to protect systems from hostile applets to protect your systems and network from potentially hostile downloaded code. In severe cases, a server can refuse to download any implementations at all.
- **Easy to Write/Easy to Use:** RMI makes it simple to write remote Java servers and Java clients that access those servers. A remote interface is an actual Java interface. A server has roughly three lines of code to declare itself a server, and otherwise is like any other Java object. This simplicity makes it easy to write servers for full-scale distributed object systems quickly, and to rapidly bring up prototypes and early versions of software for testing and evaluation. And because RMI programs are easy to write they are also easy to maintain.

- **Connects to Existing/Legacy Systems:** RMI interacts with existing systems through Java's native method interface JNI. Using RMI and JNI you can write your client in Java and use your existing server implementation. When you use RMI/JNI to connect to existing servers you can rewrite any parts of you server in Java when you choose to, and get the full benefits of Java in the new code. Similarly, RMI interacts with existing relational databases using JDBC without modifying existing non-Java source that uses the databases.
- **Write Once, Run Anywhere:** RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any Java Virtual Machine *****, as is an RMI/JDBC system. If you use RMI/JNI to interact with an existing system, the code written using JNI will compile and run with any Java virtual machine.
- **Distributed Garbage Collection:** RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network. Analogous to garbage collection inside a Java Virtual Machine, distributed garbage collection lets you define server objects as needed, knowing that they will be removed when they no longer need to be accessible by clients.
- **Parallel Computing:** RMI is multi-threaded, allowing your servers to exploit Java threads for better concurrent processing of client requests.
- **The Java Distributed Computing Solution:** RMI is part of the core Java platform starting with JDK?? 1.1, so it exists on every 1.1 Java Virtual Machine. All RMI systems talk the same public protocol, so all Java systems can talk to each other directly, without any protocol translation overhead.

Passing Behavior

When we described how RMI can move behavior above, we briefly outlined an expense report program. Here is a deeper description of how you could design such a system. We present this to show how you can use RMI's ability to move behavior from one system to another to move computing to where you want it today, and change it easily tomorrow. The examples below do not handle all cases that would arise in the real world, but instead give a flavor for how the problem can be approached.

Server-Defined Policy

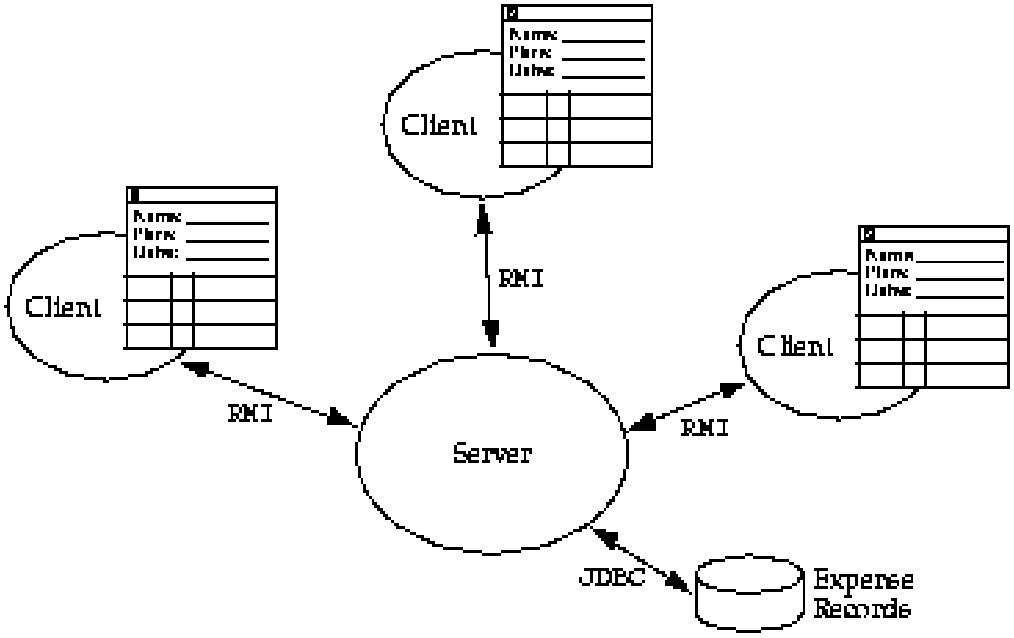


FIGURE 1 An Expense Reporting Architecture

Figure 1 shows the general picture of such a dynamically configurable expense reporting system. A client displays a GUI (graphical user interface) to a user, who fills in the fields of the expense report. Clients communicate with the server using RMI. The server stores the expense reports in a database using JDBC, the Java relational database package. So far this may look like any multi-tier system, but there is an important difference-RMI can download behavior.

Suppose that the company's policies about expense reports change. For example, today the company requires receipts only for expenses over $20. Tomorrow the company decides this is too lenient-it wants receipts for everything, except for meals that cost less than $20. Without the ability to download behavior, you have the following alternatives when designing your system for change:

- Install the policy with the client. When the policy changes, this requires updating all clients that contain the policy. You could reduce the problem by installing the client on a handful of server machines and requiring all users to run the client from one of those servers. This still would not completely solve the problem-anyone who leaves the program up and runn ing for days would not be updated, and there are always some people who copy the software to a local disk for efficiency.
- You could have the policy checked by the server when each entry is added to the expense report. This would result in a lot of traffic between client and server, clogging the

network and burdening the server. It would also make the system more fragile-a network failure would halt people in their tracks instead of only affecting them when they actually submit an expense report or start a new one. It would also mean that adding an entry would be slow, since it would require a round trip across the network to the (burdened) server.

- You could have the policy checked by the server when the report is submitted. This lets the user create a lot of bad entries which must then be reported in a batch instead of catching the first error immediately, giving the user a chance to stop making the error. Users need immediate feedback on errors to avoid wasted time.

With RMI you can have the client upload behavior from the server with a simple method invocation, providing a flexible way to offload computation from the server to the clients while providing users with faster feedback. When a user is ready to write up a new expense report, the client asks the server for an object that embodies the current policies for expense reports as expressed via a Policy interface written in Java. The object can implement the policy in any way. If this is the first time that the client's RMI runtime has seen this particular implementation of the policy, RMI will ask the server for a copy of the implementation. Should the implementation change tomorrow, a new kind of policy object will be returned to the client, and the RMI runtime will then ask for that new implementation.

This means that policy is always dynamic. You can change the policy by simply writing a new implementation of the general Policy interface, installing it on the server, and configuring the server to return objects of this new type. From that point on, any new expense reports will be checked against the new policy by every client.

This is a better approach than any static approach because:

- All clients don't need to be halted and updated with new software-software is updated on the fly as needed.
- The server is not burdened with entry checking that can be done locally.
- Allows dynamic constraints because object implementations, not just data, are passed between client and server.
- Lets users know immediately about errors.

Here is the remote interface that defines the methods the client can invoke on the server:

```
import java.rmi.*;
public interface ExpenseServer extends Remote {
```

```
    Policy getPolicy() throws RemoteException;
    void submitReport(ExpenseReport report)
        throws RemoteException, InvalidReportException;
}
```

The import statement imports the Java RMI package. All the RMI types are defined in the package java.rmi or one of its subpackages. The interface ExpenseServer is a normal Java interface with two interesting characteristics

- It extends the RMI interface named Remote, which marks the interface as one available for remote invocation.
- All its methods throw RemoteException, which is used to signal network and messaging failures. Remote methods can throw any other exception you like, but they must throw at least RemoteException so that you can handle error conditions that only arise in distributed systems. The interface itself supports two methods: getPolicy which returns an object that implements the Policy interface, and submitReport which submits a completed expense request, throwing an exception if the report is malformed for any reason.

The Policy interface itself declares a method that lets the client know if it is acceptable to add an entry to the expense report:

```
public interface Policy {
    void checkValid(ExpenseEntry entry)
        throws PolicyViolationException;
}
```

If the entry is a valid one-one that matches current policy-the method returns normally. Otherwise it throws an exception that describes the error. The Policy interface is local (not remote), and so will be implemented by an object local to the client-one that runs in the client's virtual machine, not across the network. A client would operate something like this:

```
Policy curPolicy = server.getPolicy();
start a new expense report
show the GUI to the user
while (user keeps adding entries) {
    try {
        curPolicy.checkValid(entry); // throws exception if not OK
        add the entry to the expense report
```

```
    } catch (PolicyViolationException e) {
        show the error to the user
    }
}
server.submitReport(report);
```

When the user asks the client software to start up a new expense report, the client invokes server.getPolicy to ask the server to return an object that embodies the current expense policy. Each entry that is added is first submitted to that policy object for approval. If the policy object reports no error, the entry is added to the report; otherwise the error will be displayed to the user who can take corrective action. When the user is finished adding entries to the report, the entire report is submitted.