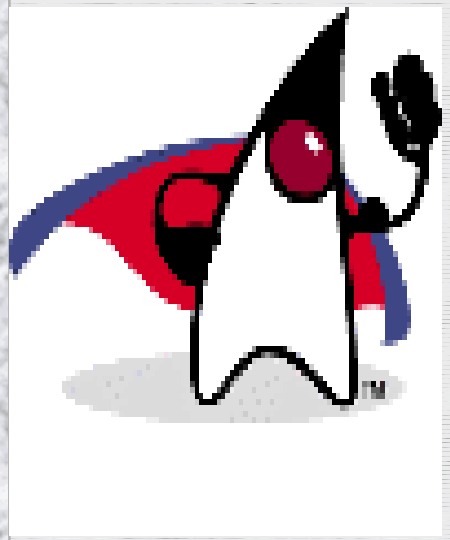
# Building Distributed Applications using JAVA - RMI



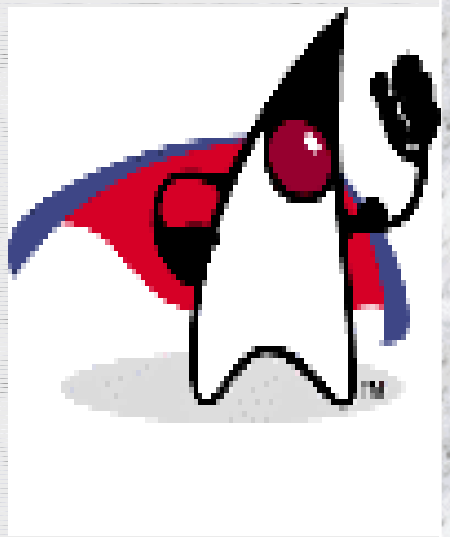

by,

Venkat

Intelligroup Inc.

(847) 292 - 8263

venkat@inforel.com

# The New World Of Computing

- Advent of the Internet and WWW
- Electronic commerce
- Distributed computing
- Component based architectures
- Off the shelf business objects
- Intelligent Agents
- Smart cards and other embedded devices

# Introduction

With the increasing growth and popularity of the Internet and networking technologies, there is an increasing demand for simple and powerful distributed applications that can be designed and maintained with a minimum of effort. There are several new distributed application technologies that are on the rise.

# Building Distributed Applications

- Sockets
- RPC (Remote Procedure Call)
- CORBA
- DCOM

- JAVA-RMI (Remote Method Invocation)

# What Do Distributed Applications Do ?

Distributed object applications need to:

- Locate remote objects
- Communicate with remote objects
- Load the code for objects that are passed around

# What is Remote Method Invocation ?

It is the action of invoking a method of a remote interface on a remote object

The method invocation on a remote object has the same syntax as the method invocation on a local object

# Java: a definition

- What is Java ?

Java is a simple, object-oriented, **distributed**, interpreted, robust, secure, architecture- neutral, portable, high-performance, multithreaded and dynamic language.

# Lets Dispel Some Java Myths

- Java is like C and C++
- Java is slow
- Java is only good for creating cool graphics on the web
- Java is easy to learn

Now On To Some Java Realities …

# The Java Platform

A platform is the hardware or software environment in which a program runs. 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. Most other platforms are described as a combination of hardware and operating system

- The Java Virtual Machine (Java VM)
- The Java Application Programming Interface (Java API)

# Java Applications and Applets

- Applications are stand alone java programs, that execute independently of a browser
- Applets are similar to applications, but they don't run standalone. Instead, applets adhere to a set of conventions that lets them run within a Java-compatible browser

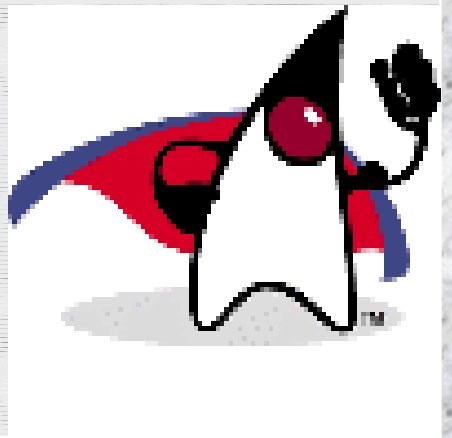
# Java - RMI

The Java Remote Method Invocation (RMI) system allows an object running in one Java Virtual Machine (VM) to invoke methods in an object running in another Java VM.

RMI provides for remote communication between programs written in the Java programming language

# WHY CHOOSE JAVA-RMI ?

- The JAVA-RMI model is simple and easy to use
- Seamless Remote Invocation on objects in different virtual machines
- Very reliable distributed applications
- Preserves the safety and security provided by the Java runtime
- Callbacks from servers to applets

# Overview Of RMI Architecture

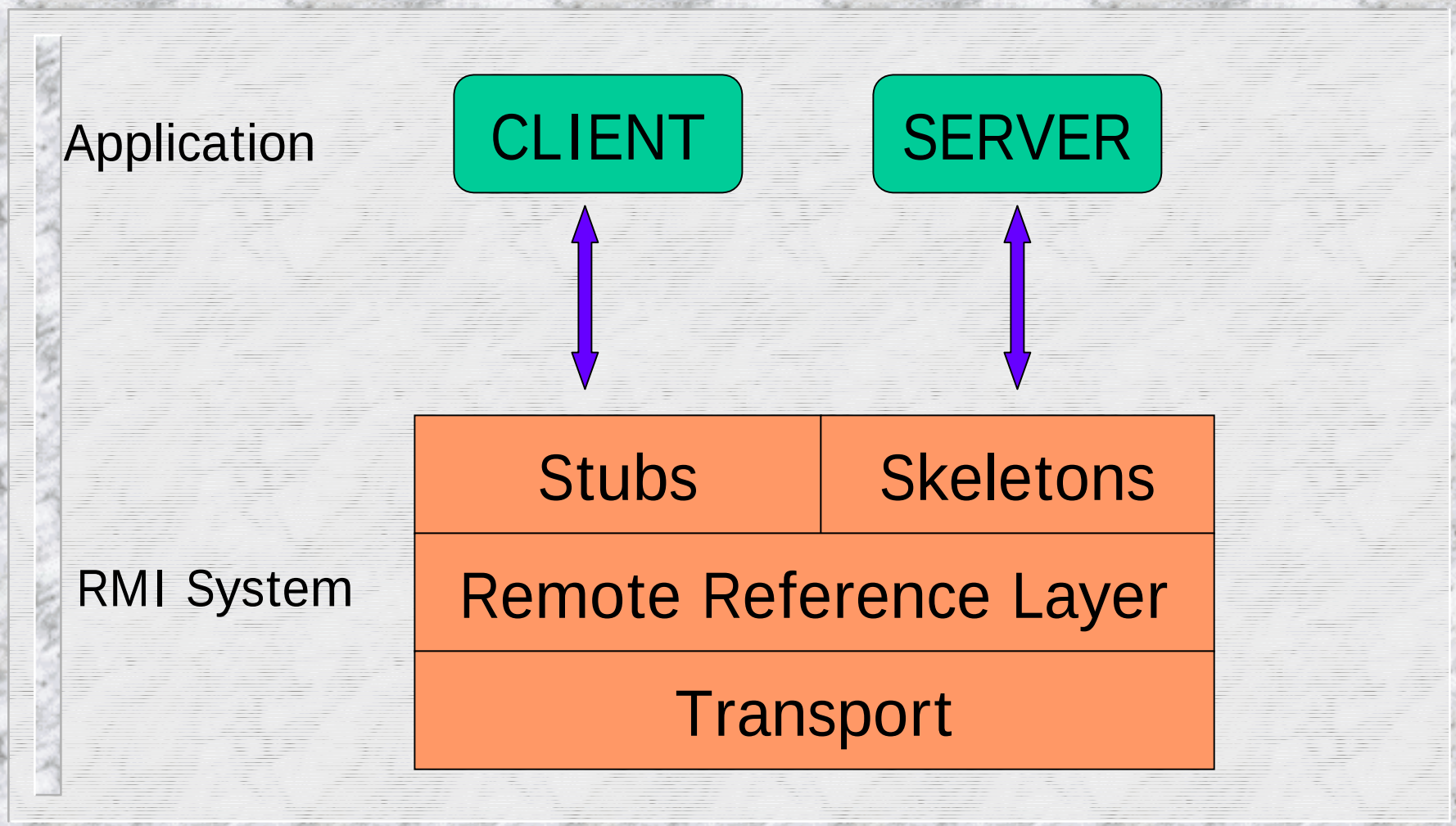
The RMI system consists of three layers:

- ***The stub/skeleton layer***
  client-side stubs and server-side skeletons
- ***The remote reference layer***
  remote reference behavior (invocation to a single or replicated objects)
- ***The transport layer***
  set up a connection, management, and remote object tracking

# Overview Of RMI Architecture

- Transparent transmission of objects from one address space to another by object serialization (java language specific)
- A client invoking a method on a remote object actually makes use of a stub or proxy for the remote object, as a conduit to the remote object
- The remote reference layer is responsible for carrying out the semantics of the invocation

# Java-RMI Architecture

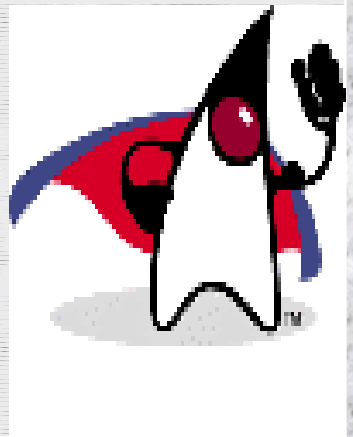


# Stubs (Client-Side Proxies)

- Initiate a call to the remote object (by calling the remote reference layer)
- Marshal arguments to a marshal stream (obtained from the remote reference layer)
- Inform the remote reference layer that the call should be invoked
- Unmarshal the return value or exception from a marshal stream
- Inform the remote reference layer that the call is complete

# Skeletons

- Unmarshal arguments from the marshal stream
- Make the up-call to the actual remote object implementation.
- Marshal the return value of the call or an exception (if one occurred) onto the marshal stream



# The Remote Reference Layer

- Responsible for carrying out specific remote reference protocol independent of stubs and skeletons
- Examples of various invocation protocols that can be carried out at this layer:
  - Unicast point-to-point invocation
  - Invocation to replicated object groups
  - Support for persistence reference to remote objects
  - Replication and reconnection strategies
- Data transmission to the transport layer via abstraction of a stream-oriented connection

# The Transport Layer

The transport layer responsibilities:

- Setting up connections to remote address spaces
- Managing connections
- Monitoring connection liveness
- Listening for incoming calls
- Maintaing a table of remote objects that reside in the address space
- Setting up a connection for an incoming call
- Locate the dispatcher for the target of the remote call and pass the connection to the dispatcher

# Transport Layer Abstractions

- ***Endpoint:*** Endpoint is the abstraction used to denote an address space or a JVM. In the implementation, an endpoint can be mapped to its transport. Given an endpoint, a specific transport instance can be obtained
- ***Channel:*** Abstraction for a conduit between two address spaces. It is responsible for managing connections between the local address space and the remote address space for which it is a channel
- ***Connection:*** It is an abstraction for transferring data (performing input/output)

# Transport Layer Abstractions

- ***Transport:*** This abstraction manages channels. Within a transport only one channel exists per pair of address spaces. Given an endpoint to a remote address space, a transport sets up a channel to that address space. The transport abstraction is responsible for accepting calls on incoming connections to the address space, setting up a connection object for the call, and dispatching to higher layers in the stream

# Thread Usage in RMI

- A method dispatched by RMI runtime to a remote object may or may not execute in a separate thread
- Some calls originating from the same client VM will execute in the same thread and others in different threads
- Calls originating from different client VM's will execute in different threads
- The RMI runtime makes no guarantees with respect to mapping remote object invocations to threads (other than the different client VM's)

# Java Distributed Object Model

- Remote Object
  an object whose methods can be invoked from another Java VM
- Remote Interfaces
  Java interfaces that declare the methods of the remote object

# Similarities of Distributed and Normal Java Object Models

- A remote object reference can be passed as an argument or returned as a result in any method invocation (local or remote)
- A remote object can be cast to any remote interface supported by the implementation
- The Java instanceof operator can be used to test the remote interfaces supported by a remote object

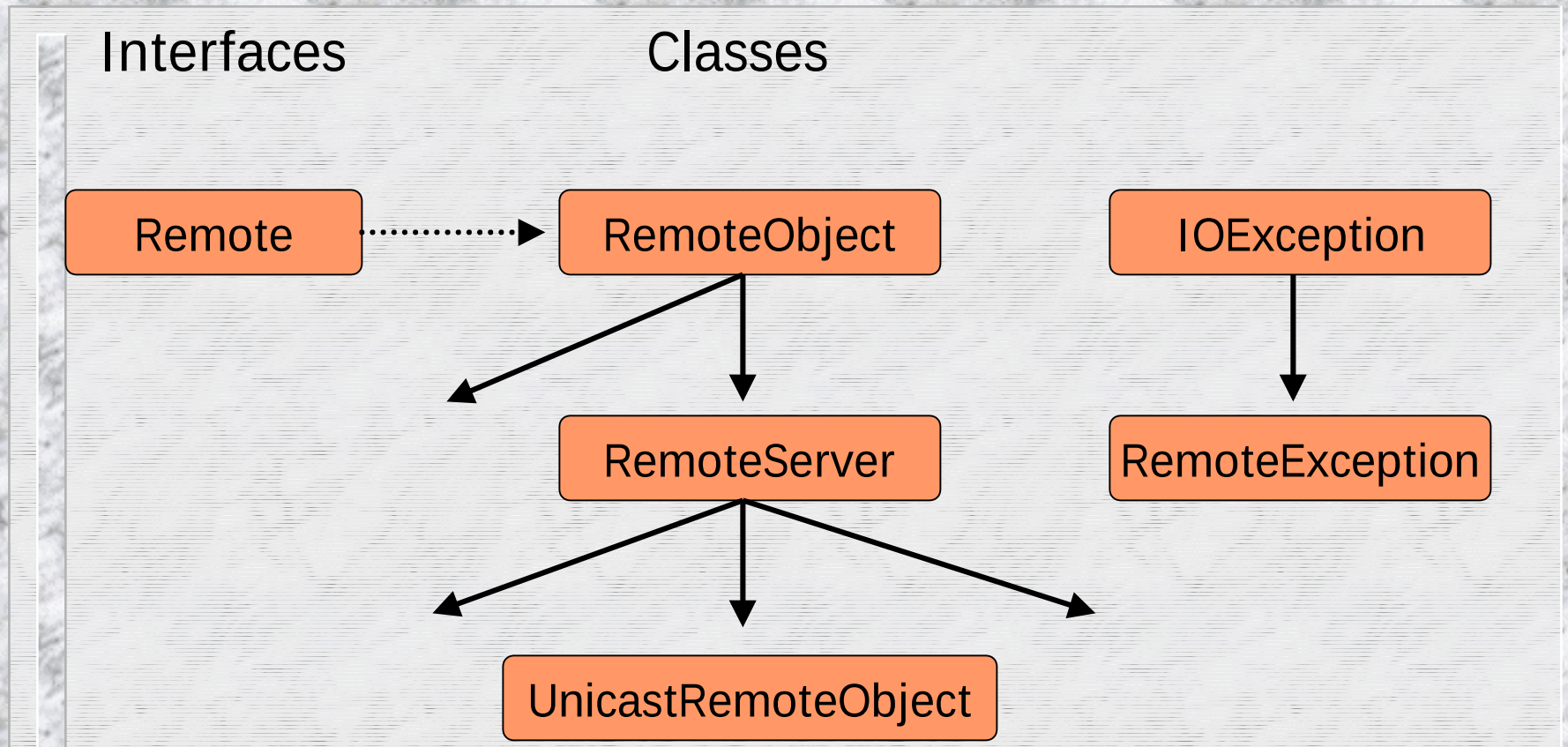
# Differences Between Distributed And Normal Java Object Models

- Clients interact only with remote interfaces and not with the implementation classes of the remote objects
- A non-remote argument is passed by copy and not by reference. A remote object is passed by reference
- Clients have to deal with additional exceptions and failure modes when invoking methods on objects remotely

# Advantages of Dynamic Code Loading

- A central and unique feature of RMI is its ability to download the bytecodes of an object's class if the class is not defined in the receiver's virtual machine
- The types and the behavior of an object, previously available only in a single virtual machine, can be transmitted to another, possibly remote, virtual machine
- RMI passes objects by their true type, so the behavior of those objects is not changed when they are sent to another virtual machine
- Allows new types to be introduced into a remote virtual machine, thus extending the behavior of an application dynamically

# RMI Interface And Classes



# The RemoteObject and RemoteServer Classes

- The java.rmi.server.RemoteObject class provides the remote semantics of Object by implementing methods for hashCode, equals, and toString
- The java.rmi.server.RemoteServer class, which is abstract, provides the methods needed to create objects and export them (make them available remotely)

# The UnicastRemoteObject Class

- The java.rmi.server.UnicastRemoteObject class defines a singleton (unicast) remote object whose references are valid only while the server process is alive
- The UnicastRemoteObject class provides support for point-to-point active object references (invocations, parameters, and results) using TCP streams

# Locating Remote Objects

- A simple bootstrap name server for storing named references to remote objects
- A remote object reference is stored using URL based methods
- Client first obtains a reference to a remote object before invoking any remote methods
- A reference to a remote object is usually obtained as a return value to a method call

# Building And Running A Distributed Application

- Define the interfaces to the remote objects
- Design and Implement the remote objects and compile the classes
- Run rmic on the compiled classes to create stubs and skeletons
- Make classes network accessible
- Start the remote registry
- Start the server … clients can invoke remote methods now !!

# Applets Vs. Applications

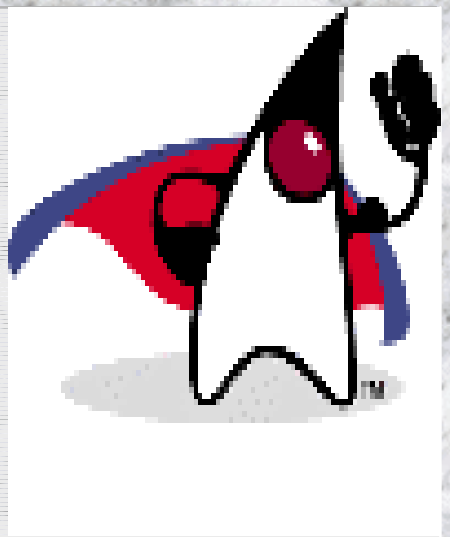
- A remote server object can extend UnicastRemoteObject and call super() in its constructor to export itself as a remote object. This is typically how an application is used
- Subclass of Applet cannot extend from another class. It can invoke the static method exportObject() available in UnicastRemoteObject and pass a reference to itself as a parameter to that method

# RMI Through Firewalls Via Proxies

- RMI transport normally opens direct socket connections to hosts on the internet
- Two alternate HTTP based mechanisms
- Enables clients behind a firewall to invoke methods on remote objects that reside outside
- RMI call embedded inside firewall trusted HTTP
- RMI call data sent as body of HTTP POST request
- Return information sent back as HTTP response
- Performance issues and limitations associated with proxies

# RMI in JDK 1.2

- Custom Sockets
- Secure Sockets
- Remote Activation
- Enhanced Garbage Collection
- Performance Increase
- *A policy file is needed*



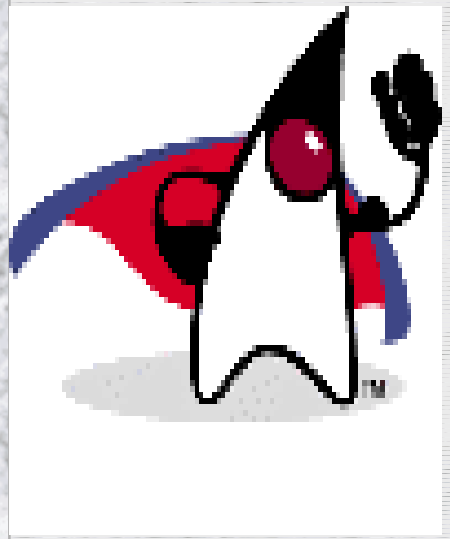
# BasketBall Game ScoreBoard Demo

- ScoreSender and ScoreReceiver
- BBServer and BBScoreBoard
- GameEvent and ShotAttempt
- CurrentScore
- BBEventGenerator
- ScoreCanvas

# Related Technologies

- Leasing
- Transactions
- Distributed Events
- Java Spaces
- Jini

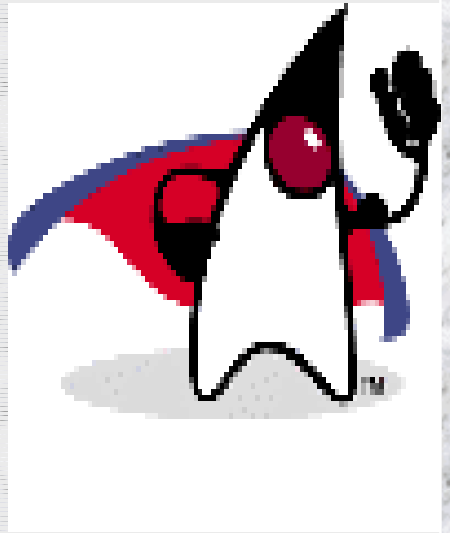
# Building Distributed Applications using JAVA - RMI




by,

Venkat

Intelligroup Inc.


(847) 292 - 8263

venkat@inforel.com

G.Venkat@Intelligroup.com