
IGATE
Speed.Agility.Imagination

Remote Method Invocation (RMI) Overview

IGATE
Speed.Agility.Imagination

Course Audience

This course is designed for Programmers

IGATE
Speed.Agility.Imagination

Prerequisites

- **Knowledge of Core Java**

IGATE
Speed.Agility.Imagination

Course Scope

This course provides:

Introduction to Remote Method Invocation (RMI)

IGATE
Speed.Agility.Imagination

Course Overview

This course will cover following topics:

- **Distributed Objects**
- **Features of Distributed Object System**
- **Distributed Object System in Market place**
- **RMI Vs CORBA**
- **Static Vs Dynamic Invocation**
- **Architecture of Distributed Object System**

Course Overview (Cont…)

- Remote Object Interactions at run-time
- RMI System Architecture
- RMI Implementation Steps
- RMI Application Deployment
- RMI Benefits

IGATE
Speed.Agility.Imagination

Distributed Objects

- Simple Idea – Objects existing on one machine (server) may be accessed from another machine through regular method call
- Eliminates need to "marshall" and "unmarshall" data sent over sockets
- Underlying socket code still exists, but is not programmed by user

IGATE
Speed.Agility.Imagination

Features of Distributed Object System

- **Object Interface Specification**
  - to allow clients to access objects
- **Object Manager**
  - The core of a distributed object system (e.g., ORB, or Registry in RMI)
  - Manages object skeletons and object references on the server
  - When a client requests a new object, the object manager locates the skeleton for the class of the requested object creates new instance based on skeleton; stores new object in the object storage sends a reference to the new object back to the client

Features of Distributed Object System

- **Registration / Naming Service**
  - Acts as an intermediary between the object client and the object manager
  - Once the interface to an object is defined, an implementation of the interface must be registered with the service so that it can be addressed by clients
- **Object Communication Protocol to handle remote object requests**
  - Must support a means of transmitting and receiving object and method references, and data in the form of objects or basic data types

Distributed Object System in Market place

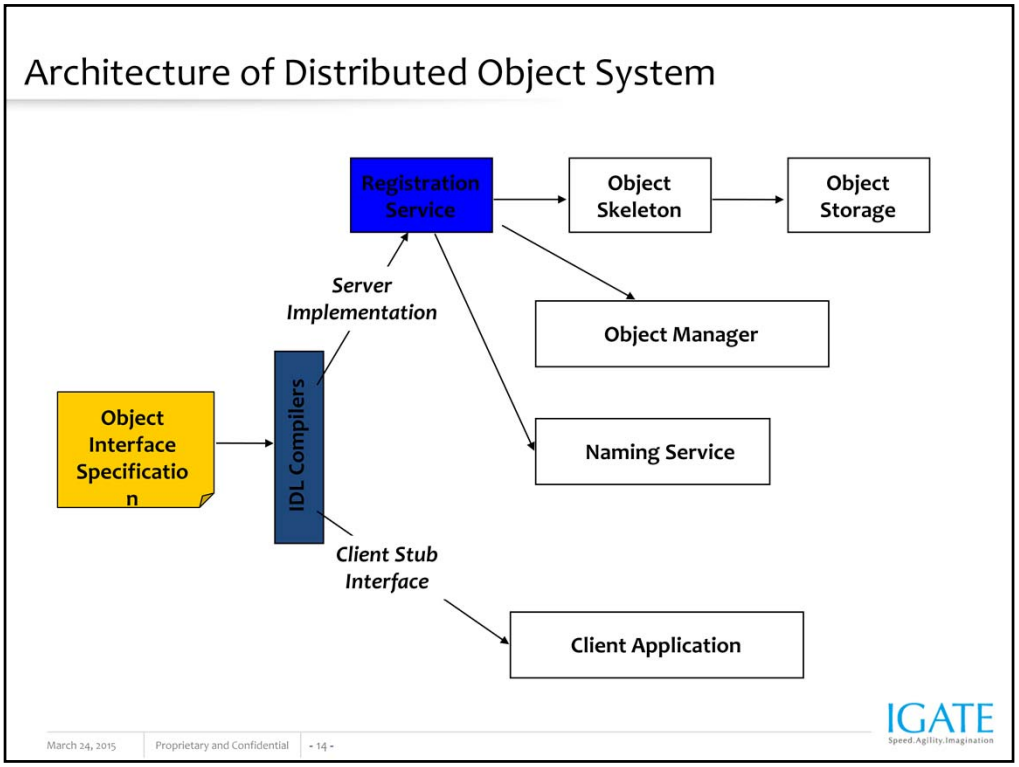
- Remote Method Invocation (RMI)
- Microsoft Component Object Model (COM/DCOM)
- CORBA (Common Object Request Broker Architecture)
- Enterprise Java Beans (EJB) [1998]
- Web services and SOAP

RMI Vs CORBA

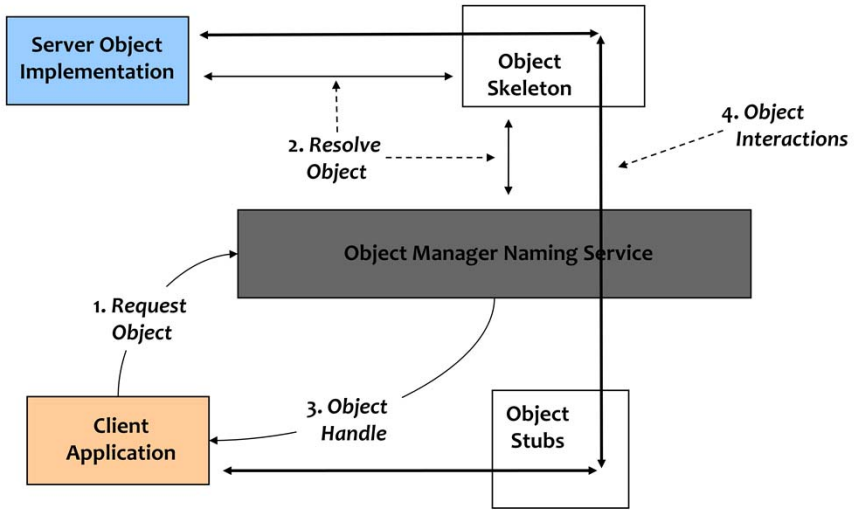
- *RMI* is Java framework for creating distributed object applications – Remote Method Invocation
- *CORBA* is alternative technology based on open standard – Common Object Request Broker Architecture
- RMI is only for pure Java applications; CORBA is language independent
- *JNI* makes this distinction a little less rigid since it allows Java to interact with other languages

Static Vs Dynamic Invocation

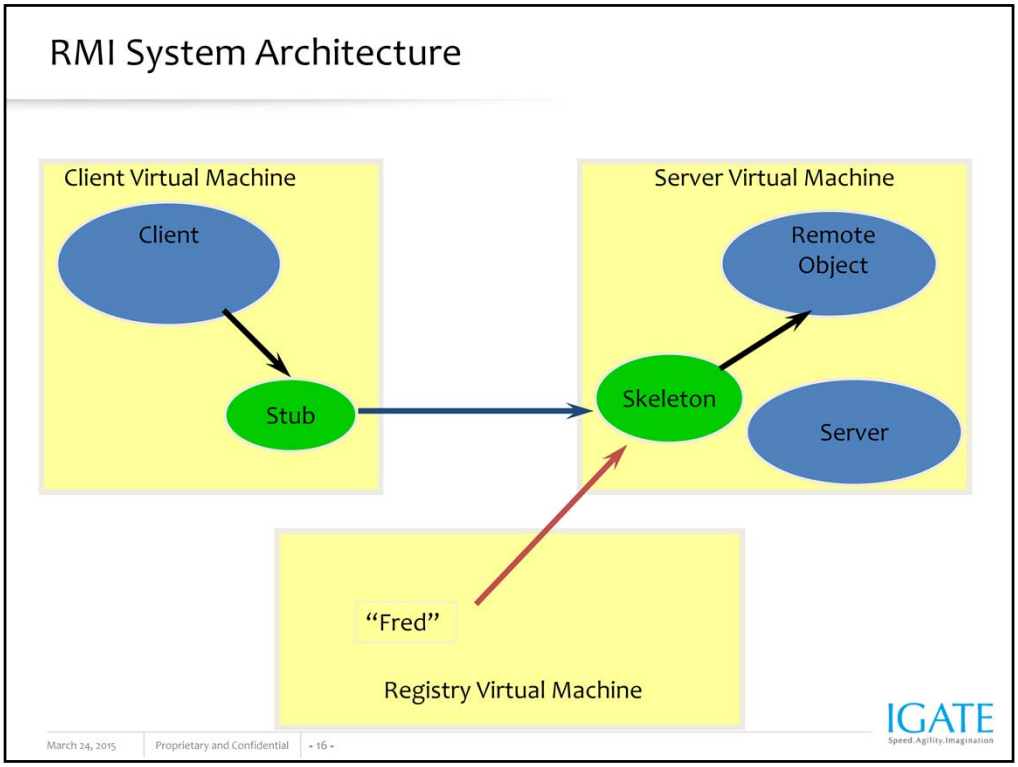
- **Static Invocation**
  - This means that the structure of the object has to be known before hand (at compile time)
  - Allows for better type checking; less runtime overhead; self-documentation
- **Dynamic Invocation**
  - Clients must discover interface-related information at runtime (e.g., using the interface repository)
  - Servers can offer new services anytime without the need for recompilation on the client side

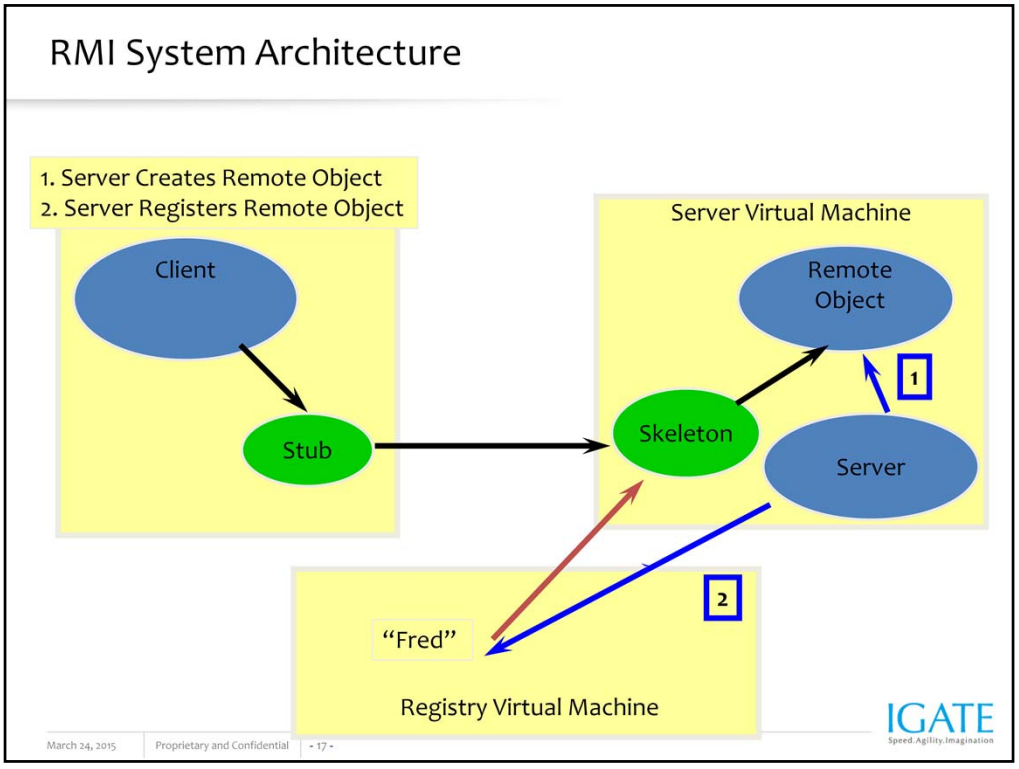

Architecture of Distributed Object System
Registration Service
Object Skeleton
Object Storage
Server Implementation
Object Manager
Object Interface Specificatio n
IDL Compilers
Naming Service
Client Stub Interface
Client Application
March 24, 2015
Proprietary and Confidential
- 14 -
IGATE
Speed.Agility.Imagination

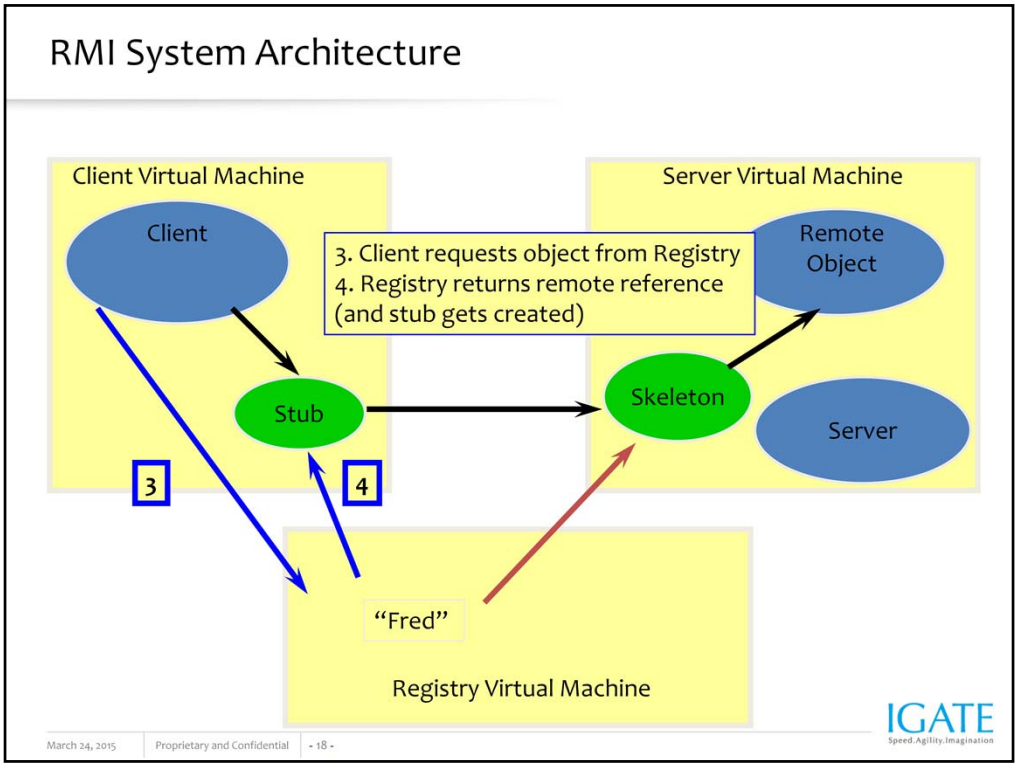
Remote Object Interactions at run-time

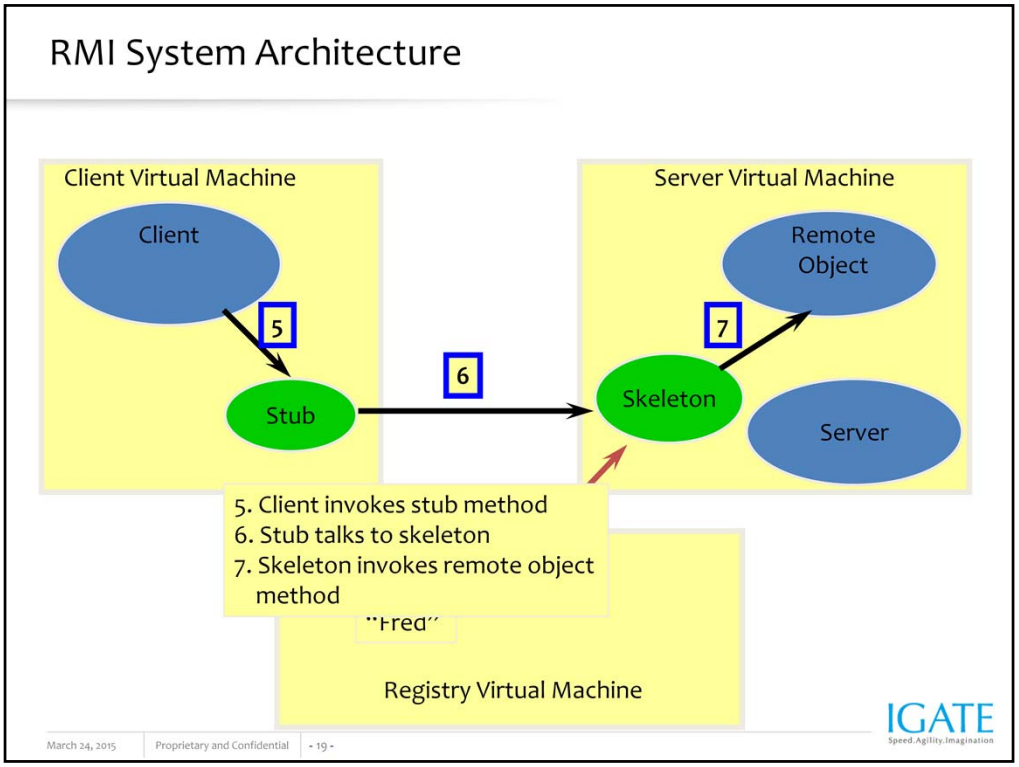


IGATE
Speed.Agility.Imagination


RMI System Architecture
Client Virtual Machine
Client
Stub
Server Virtual Machine
Remote Object
Skeleton
Server
"Fred"
Registry Virtual Machine
March 24, 2015
Proprietary and Confidential
- 16 -
IGATE
Speed.Agility.Imagination


RMI System Architecture
1. Server Creates Remote Object
2. Server Registers Remote Object
Client
Stub
Server Virtual Machine
Remote Object
1
Skeleton
Server
2
"Fred"
Registry Virtual Machine
March 24, 2015
Proprietary and Confidential
- 17 -
IGATE
Speed.Agility.Imagination


RMI System Architecture
Client Virtual Machine
Server Virtual Machine
Client
3. Client requests object from Registry
4. Registry returns remote reference
(and stub gets created)
Remote Object
Stub
Skeleton
Server
3
4
"Fred"
Registry Virtual Machine
March 24, 2015
Proprietary and Confidential
- 18 -
IGATE
Speed.Agility.Imagination


RMI System Architecture
Client Virtual Machine
Client
5
Stub
6
Server Virtual Machine
Remote Object
7
Skeleton
Server
5. Client invokes stub method
6. Stub talks to skeleton
7. Skeleton invokes remote object method
"Fred"
Registry Virtual Machine
March 24, 2015
Proprietary and Confidential
- 19 -
IGATE
Speed.Agility.Imagination

RMI Implementation Steps

- Define a remote Interface that extends java.rmi.Remote
- Define a class that implements the Remote Interface and extends java.rmi.RemoteObject or java.rmi.UnicastRemoteObject

Remote Interface Example

```
import java.rmi.Remote;

public interface Calculator extends Remote
{
public long add(long a, long b) throws java.rmi.RemoteException;
}
```

Remote Class Example

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class CalculatorImpl extends UnicastRemoteObject implements
    Calculator {
public CalculatorImpl() throws  RemoteException {  super();  }

public long add(long a, long b)throws java.rmi.RemoteException
{ return a+b;}
}
```


IGATE
Speed.Agility.Imagination

RMI Implementation Steps (Cont…)

- **Compile the Interface & the Implementation class (remote Object)**
- **Generate Stub & Skeletons using the command:**
  - Rmic –v1.2 CalculatorImpl
- **Create the server class which creates a new instance of the remote object and registers it in the registry with a unique name**
- **Create an RMI Client**

Remote server class example

```
import java.rmi.Naming;
import java.rmi.*;
public class CalculatorServer{
public CalculatorServer(){
try{
System.setSecurityManager(new RMISecurityManager());
Calculator c = new CalculatorImpl();
Naming.rebind("rmi://localhost:1099/CalculatorService",c);
System.out.println("Binding done");
}catch(Exception e){  System.out.println("Trouble: " + e); } }
```

Remote server class example (Cont…)

```
public static void main(String[] args)
{
new CalculatorServer();
}
}
```

Creating an RMI Client

- Install a a Security Manager to protect from malicious stub
- Find a registry using java.rmi.Naming
- Lookup the name of the remote Object that is registered with the registry
- Cast the return reference to the appropriate Remote Interface
- Use it!

RMI Client Example

```
System.setSecurityManager(new RMISecurityManager());
Calculator c = (Calculator)
Naming.lookup("rmi://localhost/CalculatorService");
System.out.println(c.add(4,9));
```


IGATE
Speed.Agility.Imagination

Security Issues

- Recall that client needs auto-generated up-to-date stub functions.
- If these are available locally on the client, there is no security issue.
- However, keeping a local installation can be cumbersome. Often stubs are downloaded via other servers (we'll see how to do this).
- In this case, a SecurityManager needs to be installed to ensure that the stubs are not hostile (unless applet is used, which has its own SecurityManager).

RMISecurity Manager

- **Easiest way to do this is to use java.rmi.RMISecurityManager as:**
  System.setSecurityManager(new RMISecurityManager());
- **This by default restricts all code from making socket connections.**
- **Obviously this is too strict. Need a *policy file* to allow client to make network connection to rmi port. It would look something like:**

```
grant{ permission java.net.SocketPermission
        "*:1024-65535", "connect"}
java Client –Djava.security.policy=client.policy
```

RMI Implementation Steps (cont…)

- **Create the java policy file or make the following entry in the java.policy file :**

Grant

{

.

.

Permission java.security.Allpermission “”,””

}

RMI Implementation Steps (cont…)

- **Open 3 consoles**
  - In first console run rmiregistry - it will need the Interface & the stub
    - start rmiregistry
  - In second console run the server – it will need the Interface & remote class
    - java CalculatorServer
  - In the third console run the client – it will need the Interface & stub
    - Java CalculatorClient

RMI Application Deployment

- Very simple if client/server both have up-to-date copies of all class files
- However, this is unrealistic and impractical.
- Better if client can load dynamically load classes remotely.
- RMI provides such a mechanism built on top of standard servers.

Deployment (cont…)

- ***For server, following classes must be available to its classloader:***
  - Remote service interface definitions
  - Remote service implementations
  - Stubs
  - All other server classes
- ***For client***
  - Remote service interface definitions
  - Stubs
  - Server classes for objects used by the client (e.g. return values)
  - All other client classes

RMI Benefits

- **Enables use of Design Patterns**
  - Use the full power of object oriented technology in distributed computing, such as two- and three-tier systems (pass behavior and use OO design patterns)
- **Safe and Secure**
  - RMI uses built-in Java security mechanisms
- **Easy to Write/Easy to Use**
  - A remote interface is an actual Java interface
- **Distributed Garbage Collection**
  - Collects remote server objects that are no longer referenced by any client in the network


Thank You
March 24, 2015
Proprietary and Confidential
- 35 -
IGATE
Speed.Agility.Imagination