Remote Procedure Call (RPC) et al.
Distributed Objects

- **Object has**
  - Interface (abstract type)
  - Implementation (concrete type)
  - Local reference, e.g., value of a monotonically increased counter, memory address

- **Remote object has**
  - Interface for remote invocations
    - Described in the implementation language: 1st class RMI
    - Otherwise: described in a separate language
  - Usually local interface limited to access inside the process
  - Implementation
  - Global reference, e.g., (host id, process id, obj id)
Interaction Scheme
Stubs and Skeletons

Client side: stub/proxy
- Offers same interface as server object: mimics the server
- Usually bound to a single server
- Marshals the request into a stream of bytes
  - Method id (e.g., name)
  - Arguments
- Additional features:
  - Caching of values
  - Load balancing
  - Statistics
  - ...

Server side: skeleton
- Represents the server object
- Bound to a single server
- Usually several proxies for a server
- Unmarshals the request and calls the corresponding method on the server object
- Additional features:
  - Persistence
  - ...

Invocations

- Transformed to messages, and sent to the « other side » (*marshaling*)

The « other side »: *skeleton*

- Server-side counterpart to the stub
- Extracts request arguments from message (*unmarshaling*) and invokes the server object
- Marshals return value and sends it to the invoker side, where stub unmarshals it and returns the result to invoker
- **Skeleton type**
  - Delegation: skeleton is separate object (of arbitrary type)
    - Associated with the effective server object (*binding* is usually made by application)
  - Inheritance: a supertype of the server object type
    - Developer subclasses a skeleton class generated for the server object interface, e.g.,
      ```java
      public class BobSkeleton implements Bob {}
      public class BobServer extends BobSkeleton {
          ...
      }
      ```
Marshaling

- Unbound objects: serialized and passed by value
- Primitive types: ditto
- Bound objects: passed by reference

Mainly bound objects remotely « visible »

Stub creation

- Received as argument/result of remote method invocation, or
- From lookup service
Anthropomorphism

Video conference?
- Not really - participant is very aware that she/he is talking « through » a computer

Sci-Fi scenario: hologram
- Participant talks to hologram which represents distant participant
- Participant does not see difference between hologram and real person
Further Concepts

Repositories

- Reference Repository
  - *Find* new remote *objects* (locate objects, i.e., bootstrapping)
- Interface Repository
  - *Discover* new remote object *types* (browse remote types)
- Object Repository
  - *Initialize* new remote *objects* (automatic server activation)

Advanced concepts

- Dynamic invocations
- Interception
- Threading
Remote Procedure Call (RPC) et al.

Java Remote Method Invocation (Java RMI)
Java RMI: At a Glance

- Allows distributed Java objects to interact
  - Through (remote) method invocations, since Java 1.1
  - Invocations are synchronous (even if no reply)

- 1st class RPC package
  - Fully integrated with Java language
  - Remote interfaces are described through Java interfaces

- Separate compilation
  - Generate stubs and skeletons according to interfaces
  - Compile application

- As of 1.5: dynamic proxies are preferred
Java RMI Architecture
Stub/Skeleton Layer

**Stub**
- Has same interface as remote object
- Initializes call to remote object
- Marshals arguments to stream
- Passes stream to remote reference layer
- Unmarshals the return value
- Informs the remote reference layer that call is complete

**Skeleton**
- Unmarshals arguments from the stream
- Makes (up-)call to the remote object implementation
- Marshals the return value or an exception onto the stream
Remote Reference Layer

- Carries out remote reference protocol
  - Independent of stubs/skeletons

- Remote object implementation chooses invocation protocol
  - Unicast point-to-point
    - Replicated object group
      - Support for specific replication strategy
  - Support for persistent reference to remote object (automatic activation of remote object)
  - Reconnection strategies
Transport Layer

 Responsibilities

- Connection set-up to remote address space
- Managing connections
- Monitoring connection « liveness »
- Listening for incoming calls
- Maintaining a table of remote objects
- Connection set-up for incoming call
- Locating the dispatcher for the target of the remote call

 Abstractions

- *Endpoint:* denotes an address space or JVM
- *Channel:* conduit between two address spaces, manages connections
- *Connection:* data transfer (input/output)
- *Transport:* Manages channels
Design a Java RMI Application

1. Write the interfaces of the remote (i.e., remotely accessible) objects: coarse grain
2. Write the implementations of the remote objects
3. Write other classes involved: fine grain
4. Compile the application with javac
5. Generate stubs and skeletons with rmic
Declaring a Remote Interface

- Objects are remotely accessible through their remote interface(s) only
- Methods to be exported are declared in an interface that extends the `java.rmi.Remote` interface

**Remote interfaces**
- Must be `public`
- All methods must declare `java.rmi.RemoteException` in the `throws` list: represent exceptions due to distribution
import java.rmi.*;

public interface Hello extends Remote {
    public void print() throws RemoteException;
}

Implementing Remote Interface

- Implement the Remote interface

  ✓ Abstract class `java.rmi.server.RemoteObject` implements `Remote`
    - Remote behavior for `hashCode()`, `equals()` and `toString()`

  ✓ Abstract class `java.rmi.server.RemoteServer` extends `RemoteObject`
    - Functions to export remote objects
Concrete class

✓ `java.rmi.server.UnicastRemoteObject` extends `RemoteServer`
  ● Non-replicated remote object
  ● Support for point-to-point active object references (invocations, parameters, and results) using TCP
  ● Inheritance: subclass `UnicastRemoteObject`

Note

✓ Own exceptions must not subtype `RemoteException`
A Hello World Implementation

import java.rmi.*;
import java.rmi.server.*;

public class HelloImpl extends UnicastRemoteObject
    implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public void print() throws RemoteException {
        System.out.println("Hello World");
    }
}
Simple Classes and Interfaces
Constructing a Remote Object

- **The Constructor**
  - Calls the no-argument constructor of the `UnicastRemoteObject` class (implicitly or explicitly)
  - Which exports a `UnicastRemoteObject`, meaning that it is available to accept incoming requests by listening to calls from clients on an anonymous port
  - Throws `RemoteException`, since the constructor of `UnicastRemoteObject` might do so, if the object cannot be exported
    - Communication resources are unavailable
    - Stub class cannot be found, ...

- **Alternative: Delegation**
  - Explicitly export the object
    `UnicastRemoteObject.exportObject()`
public class HelloServer {

    public static void main(String[] args) {
        ...  
        Hello hello = new HelloImpl();  
        // Register object (e.g., naming service)  
        // What’s up doc?
        ...
    }
}

Starting a Server
public class HelloClient {

    public static void main(String[] args) {
        
        // Lookup object (e.g., naming service)
        Hello hello = ...;

        // Invoke the remote object
        hello.print();

        // That's all folks...
    }
}

Parameters and Return Values

- Can be
  - Local objects
    - Primitive types (e.g., `int`, `boolean`, ...)
    - Serializable, i.e. implementing `java.io.Serializable`:
      fields are copied (except `static` or `transient`) and serialized
  - Remote Objects
    - Passed by reference

- Serialization
  - Can be overridden
  - A copy of the object is created at the client side
Exceptions

- All RMI exceptions subclass `RemoteException`

- Examples
  - `StubNotFoundException` or `SkeletonNotFoundException`: forgot `rmic`?
  - `ServerError`: error while server executes remote method
  - `ServerException`: remote exception in server’s remote method call, received by 1st invoker
Distributed Garbage Collection

- **(Local) object garbage collected if**
  - No (local) reference to it exists

- **Remote object garbage collected if**
  - No local reference to it exists
  - No remote reference to it exists

- **JVM keeps track of the # of refs to an object**
  - First reference triggers message send to JVM hosting the object
  - Discarding last reference triggers message send as well
  - When no remote JVM references object anymore, reference becomes « weak »
More Information

- Java site: [http://java.sun.com](http://java.sun.com)
- Tutorial: [http://java.sun.com/j2se/1.5/docs/guide/rmi/](http://java.sun.com/j2se/1.5/docs/guide/rmi/)