

A Distributed Object Model for the Java System

Ann Wolrath, Roger Riggs, and Jim
Waldo

- Communication mechanisms are key in distributed systems – need a good model
 - sockets
 - can be complicated to encode/decode messages
 - RCP – procedure calls,
 - but in Java method invocations on objects
 - CORBA – assumes heterogeneous platforms, so uses IDLs to generate language neutral objects that don't match Java object model
 - but in Java already have homogeneous JVM
- Java model: Remote Method Invocation on remote objects for Java and JVM

Goal – Language integration

- Enable distributed computing and communication but maintain language-specific (Java) object semantics *as much as possible*
 - similar interfaces as if object local, preserve some behavior
 - keep it simple: expose exceptions related to distributed interaction

Goal – System Integration

- Use safety, runtime class loading, security, garbage collection... already provided by JVM
- Support different invocation semantics
 - unicast, multicast... (like with Subcontracts in Spring)
- Support different reference semantics
 - persistent, non-persistent references to remote objects
- Multiple transports for pair of endpoints
 - UDP, TCP... (like in Network Objects)

Language Integration -- Distributed Object Model

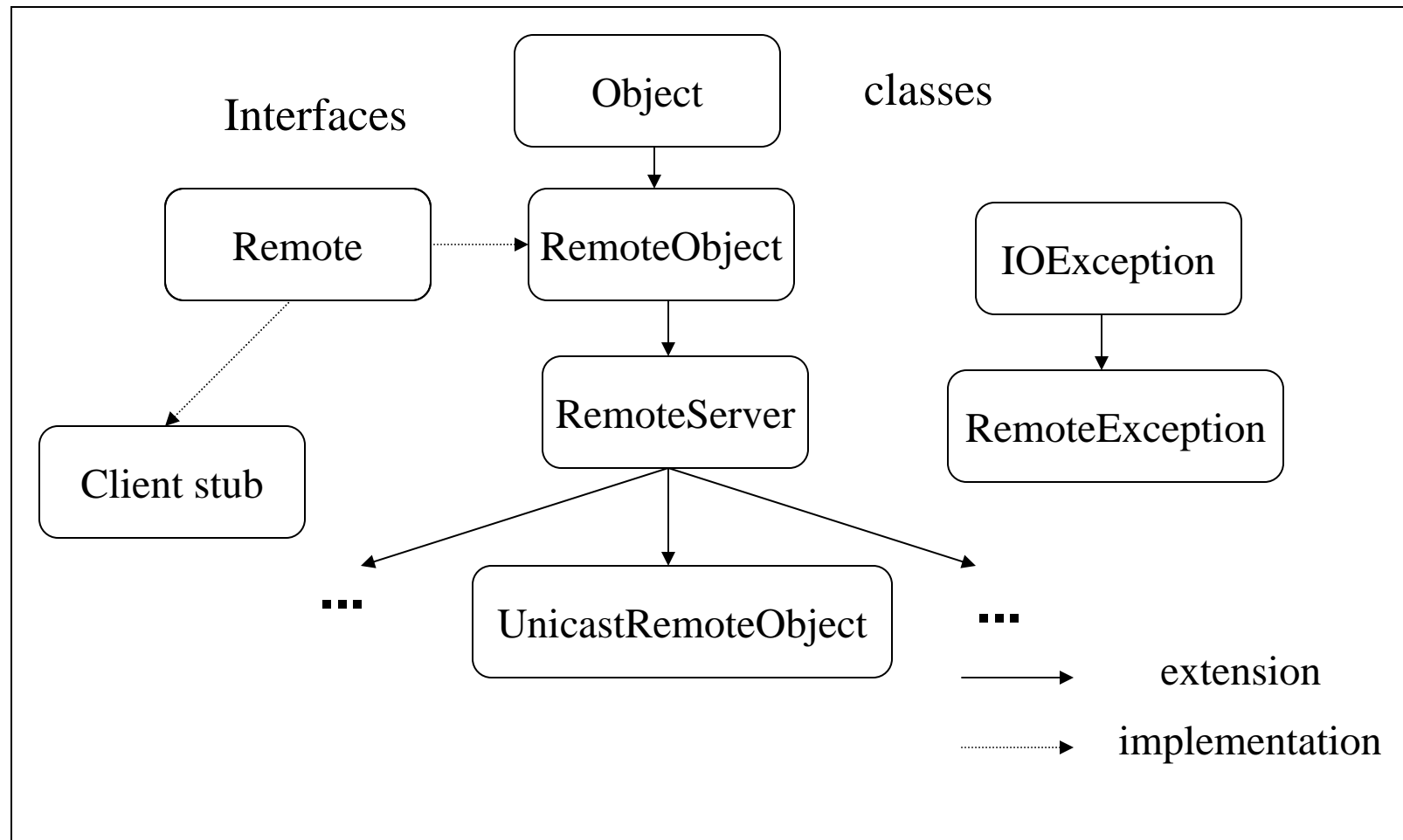
Java Object model

- Object
- Interface
 - declare a set of methods for Java object without implementation
- Method Invocation
 - primitive type passed by value
 - object passed by reference

Distributed Object model

- Remote object
 - object whose methods can be accessed from another address space
- Remote interface
 - an interface that declares the methods of a remote object
 - throws Remote Exception to deal with different failure models
- Remote Implementation
 - export object implementation to RMI runtime (Remote Server)
- RMI
 - non-remote object passed by value
 - remote object passed by reference

RMI Interfaces and Classes



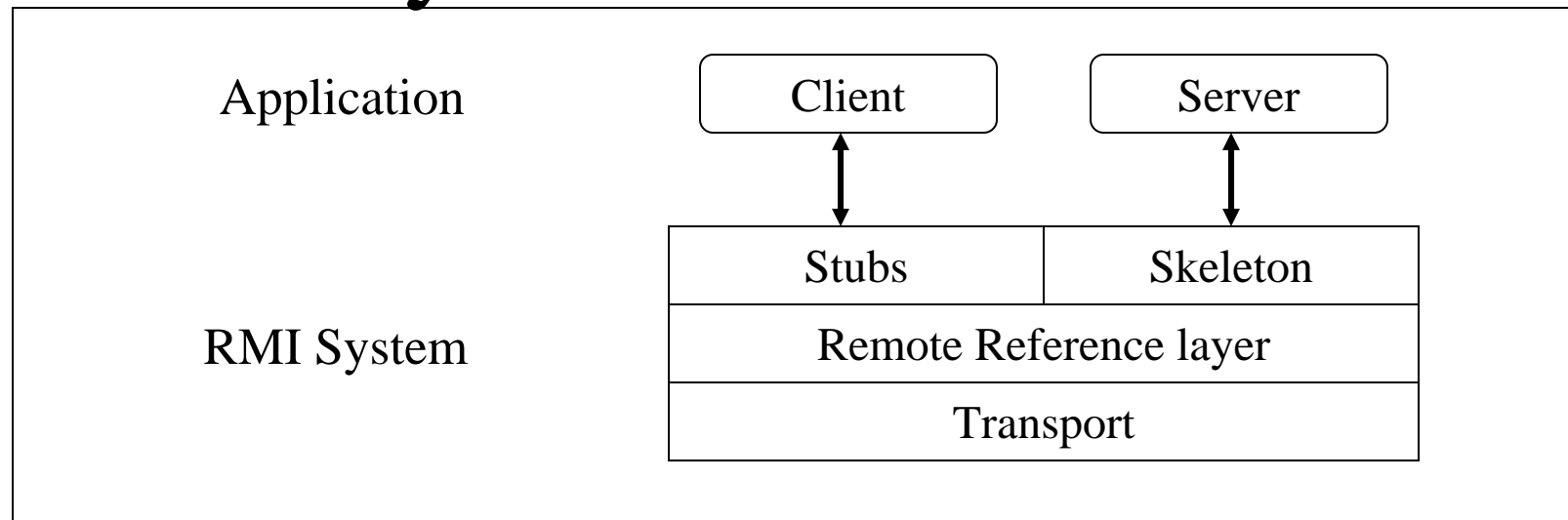
RMI details

- Remote reference type
 - client stubs have same remote interface as the remote object, therefore Java treats them as objects with same type
- Remote method invocation
 - same syntax, but exceptions have to be caught
 - exception tells only that *something* happened

More details...

- Some semantics have to change
 - client has a reference to remote object
 - methods of class Object (equal, hashCode...) don't involve a remote call, so can operate on reference...
 - some methods don't make sense, e.g. cannot use wait/notify for synchronization
- Registry for locating objects

System Architecture



- Stub/Skeleton layer – client proxy and server dispatcher
- Remote reference layer – invocation and reference semantics
- Transport layer – actual connection setup and management

Pickling

- Used for object transmission across address spaces
 - Stubs/skeletons marshal/demarshal ‘pickled’ streams
 - When marshalling remote object, marshal stream embeds URL info of the stub code
 - On demarshalling, marshal stream loads stub code if it’s not available locally

More System Integration

- Distributed Garbage Collection
 - RMI runtime keeps count of all references to an object within all JVMs.
- Dynamic Stub loading
 - specialized class loader (can load exact class)
 - security manager