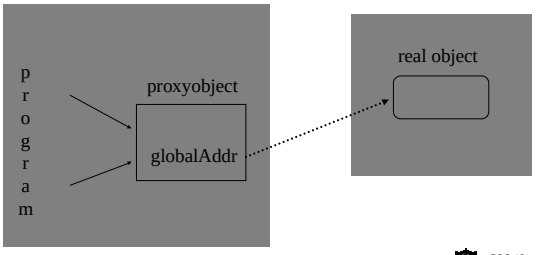
Network Objects

- first good implementation: DEC SRC Network Objects for Modula-3
- recent implementation: Java RMI (Remote Method Invocation)
- idea: allow programs to treat remote objects just like local objects
  - remote references
  - remote invocation

Remote References

- naming a remote object ("global address")
  - machine (IP address)
  - port number
  - (boot time)
  - unique object ID number
- works in protocol, but not for a Java program
- for program, use stand-in proxy object

Proxy Objects



Proxy Objects

- separate proxy object for each remote object that is referenced
- proxy object looks just like real object
  - implements the same Java interface
- proxy's methods do RPC to real object
  - handle arguments, return values, exceptions correctly
- proxy code generated by RMI compiler

Proxy Types

- remote objects declared as having an interface type
  - no variables, only methods
- proxy implements same interface as real object
- remote interface types must extend java.rmi.Remote interface

Proxy Table

- each process keeps a table of all proxy objects
  - maps global address to proxy object
- use table to maintain one-to-one mapping from proxy object to remote object
- interactions with garbage collection (later)

Remote Invocation

- when a proxy method is invoked:
  - invocation message sent to remote process
    - » contains global address of object, method ID, arguments
  - remote process invokes real object
  - return message sent back to proxy object
    - » contains return value or exception
  - proxy method call returns correct value, or throws correct exception

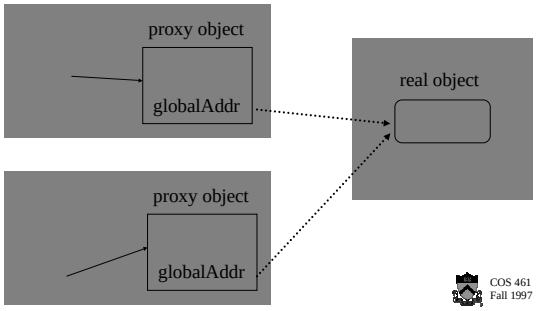
RMI Service Code

- runs in each process that exports objects
- waits for incoming connections
- accepts requests, passes them to per-class request handlers
- request handler translates global address to local reference, calls method, gets return value or catches exception, sends reply
- request handler code generated by RMI compiler

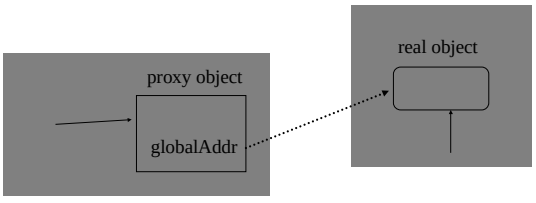
Passing Arguments

- primitive types (int, boolean, etc.) and ordinary objects passed by copy
- Remote objects passed by reference
  - send across global address
  - at destination, translate into
    - » pointer to existing proxy object, if there is one
    - » pointer to new proxy object, otherwise
- remaining question: how to copy objects

Passing a Remote Reference (1)



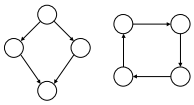
Passing a Remote Reference (2)



Copying Objects

- use ObjectOutputStream, ObjectInputStream
- write out fields one at a time
- if field is an Object type, recurse

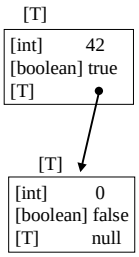
two tricky cases



Copying Objects

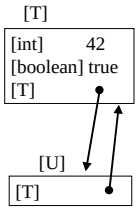
- output stream is a sequence of objects
  - sequence number on each object in stream
- to send a reference to an object, send the corresponding sequence number
- keep table mapping pointer to sequence number
  - add entries when new objects are sent

Example (1)



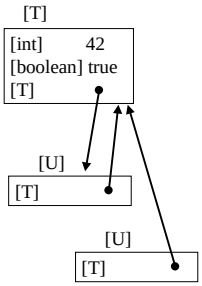
[class] <code for T>
[int]
0 0 0 42
[boolean]
1
[reference]
2
[class] <code for T>
[int]
0 0 0 0
[boolean]
0
[null]

Example (2)



[class] <code for T>
[int]
0 0 0 42
[boolean]
1
[reference]
2
[class] <code for U>
[reference]
1

Example (3)



[class] <code for T>
[int]
0 0 0 42
[boolean]
1
[reference]
2
[class] <code for U>
[reference]
1
[class] <code for U>
[reference]
1

Class Codes

- two parts to class code
  - "unique" ID number
    - » hash of class name, variable names and types, method names and signature
  - URL of Java bytecode for class
    - » allows recipient to load code, if it wants
      - (possible security problem)

Java Serialization

- generalizes argument-passing to allow any object to be turned into a byte-array, and vice versa
- to use, declare object as implementing Serializable
- details
  - "transient" fields ignored
  - can customize serialization/deserialization
  - versioning support

Dealing with Failure

- What if net fails, or object's owner crashes?
  - proxy object detects failure
  - proxy throws RemoteException

Using RMI

- write interface (extends Remote)
- write server class (extends UnicastRemoteObject, implements interface)
- run stub compiler (rmic) on server class
  - creates glue classes, one for each side
- write code that uses class, compile and run

Garbage Collection

- ideal situation: extend Java's garbage collector to "do the right thing" in the distributed case
  - very difficult and inefficient in practice
- lesser goal: collect everything, except for non-local cycles of garbage

Garbage Collection

- collecting proxy objects
  - when no more local references to proxy, garbage-collect proxy
    - » done "automatically" by existing garbage-collector
  - afterward, inform remote object that we don't hold a proxy any more

Garbage Collection

- collecting remote objects
  - RMI support code keeps track of who has proxies to the object
  - support code pings proxies periodically to make sure they're still alive
  - support code keeps a reference to the object as long as there are any proxies
  - existing garbage-collector unmodified

Finding Objects

- Where do remote references come from?
  - passed or returned by remote calls
- Where does the first one come from?
  - special "registry" object on each host
  - special call to get a reference to a Registry object
    - » uses a hack

Security?

- short answer: questionable
- long answer
  - messages not encrypted or signed
    - » modification of calls
    - » unauthorized calls
    - » corruption of garbage-collection algorithm
  - serialization allows illegal object tampering
- challenging to fix!