Question 3: RMI + Proxy pattern

IT-SDJ2-A21

Software Engineering

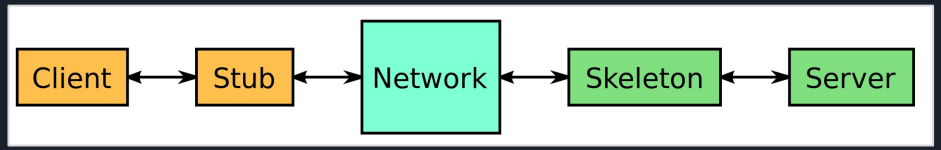
VIA University College

Jordi Lazo

RMI

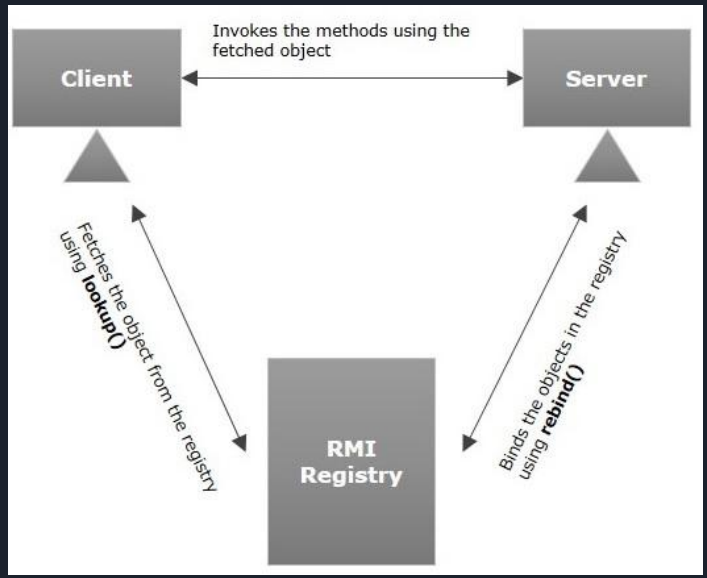
Main parts of a communication between computers using RMI

- RMI is an API that provides provides remote communication between Java programs. The RMI allows an object to invoke methods on an object running in another JVM.
- RMI provides remote communication between the applications using two objects *stub* and *skeleton*.
- Stub → is a representation of the remote object at client. It resides in the client system; it acts as a gateway for the client program.
- Skeleton → object which resides on the server side. stub communicates with this skeleton to pass request to the remote object.



Remote interface and registry

- RMI registry is a namespace on which all server objects are placed.
- Each time the server creates an object, it registers this object with the RMIregistry (using bind() or reBind() methods). These are registered using a unique name known as bind name.
- To invoke a remote object, the client needs a reference of that object. The client fetches the object from the registry using its bind name (using lookup() method).
- *Remote interface* is an interface that declares a set of methods that may be invoked from a remote Java virtual machine



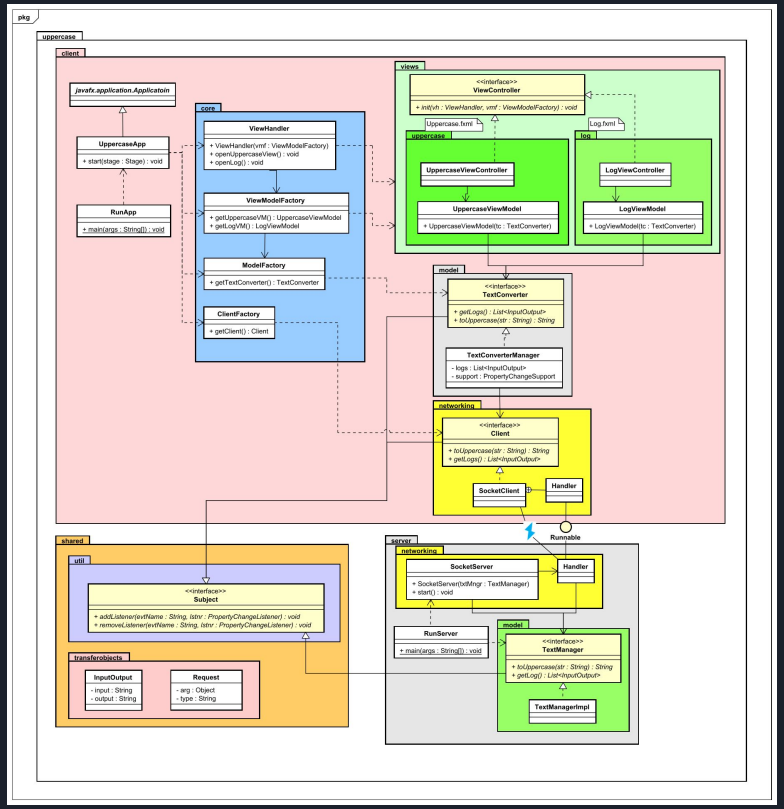
Server call-back to a client

- RMI callback occurs when the client sends a remote reference to another service (server), and the server calls methods on the client's reference whenever it is needed.
- RMI call-back example, a temperature service is implemented. The server supplies information about the temperature. The temperature monitor (client) registers with the temperature service (server). When the temperature does change, a callback is made, notifying registered client references.

Server broadcast to all clients

1. The clients needs to export a remote object that is used for callbacks.
2. Each client needs to register its callback with the server.
3. The server needs to maintain a collection of these callbacks.
4. The server needs to iterator over the collection calling each callback.

UML + Java example



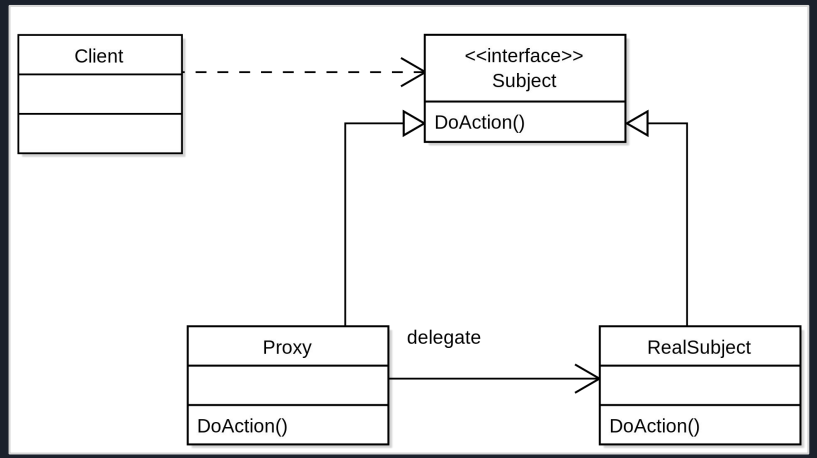
Proxy pattern

What is the purpose?

- Proxy pattern is a structural design pattern that provide a substitute or placeholder for another object.
- A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.
- A class represents functionality of another class.
- A object is created having the original object to interface its functionality to outer world.
- The access to an object should be controlled.

What are the different parts involved? How do they interact?

- The *Client* object works through a *Proxy* object that controls the access to a *RealSubject* object. In this example, the *Proxy* forwards the request to the *RealSubject*, which performs the request.
- To act as substitute for a subject, a proxy must implement the *Subject* interface. Clients can't tell whether they work with a subject or its proxy.



UML + Java example