Middleware Technologies and RMI

1 Middleware Technologies

User perspective:

- Hide users from the implementation details and the ”distributed” nature

Developer perspective:

- Hide DS developers from low-level details
- Provide common programming abstraction and infrastructure for constructing distributed applications

2 Examples of middleware

Service broker:

- Serve as the prime interface for accessing a distributed system
- Identify or discover suitable remote component to serve the purpose
- Provide a variety of distributed system features, such as concurrency control, load distribution and fault tolerance

3 Developing a distributed system with middleware

Aims:

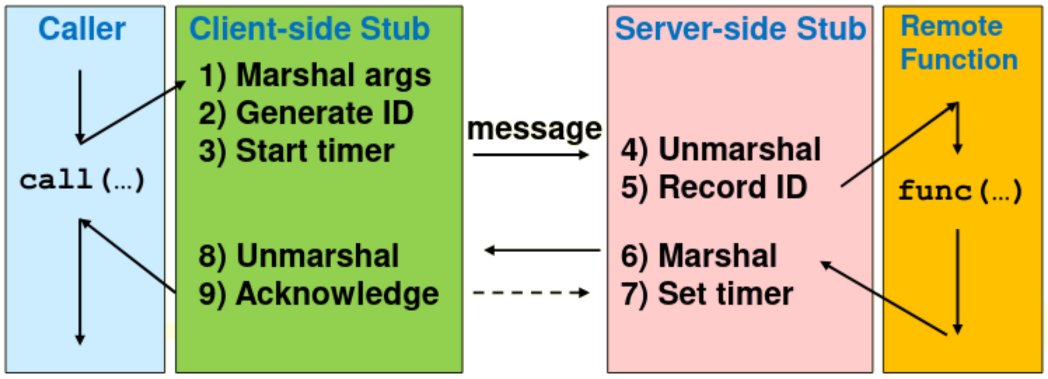
- Focus on application logics
- Avoid implementing socket communication and dealing with network architecture and its dynamic changes

Middleware options:

- Remote Procedure Call (RPC) - Inter-process communication; execute a remote function without the programmer coding the network communication
- Object-Oriented Middleware (OOM) - Extend the idea of RPC to allow remote invocation of objects
- Message-Oriented Middleware (MOM)
- Web Services

3.1 Remote Procedure Call (RPC)

- Adapt traditional program paradigm, dividing a system into functions
- Mask remote function calls as being local, supporting distributed development



- Request/reply paradigm usually implemented with message passing in RPC service
- Marshalling of function parameters and return value

3.1.1 RPC Program development

- Server - Defines the service interface using an interface definition language (IDL), which specifies names, parameters, and types for all client-callable procedures
- Stub compiler - Reads the IDL declarations and produces two stub functions for each server function (server-side and client-side)
- Linking - Server programmer implements the service's functions and links with the server-side stubs. Client programmer implements the client program and links it with client-side stubs
- Operation - Stubs manage all of the details of remote communication between client ans server

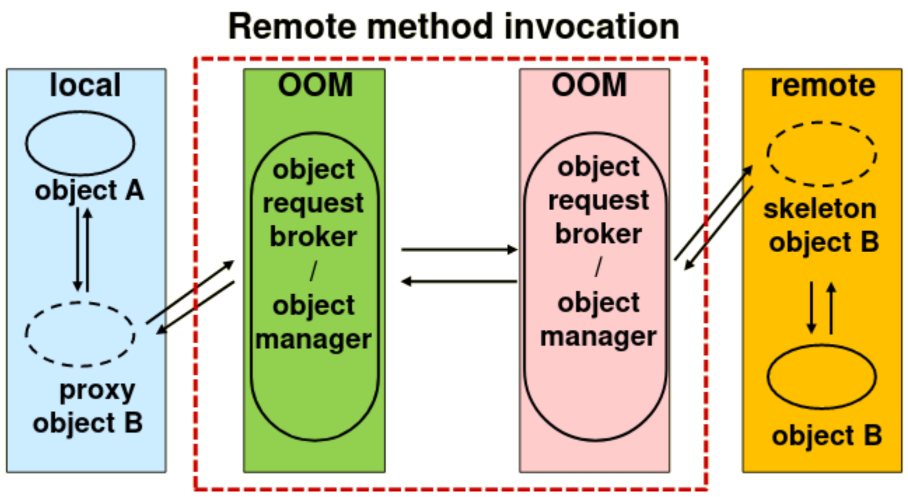
3.1.2 Properties and Limitations

Synchronous request/reply interaction:

- Holds a connection open and waits until the response is delivered or the timeout period expires
- Tight coupling between client and server
- Client may block for a long time if server loaded
- Slow/failed clients may delay servers

Program paradigm: not object-oriented - invoke functions on servers - no encapsulation or inheritance support

3.2 Object-Oriented Middleware (OOM)

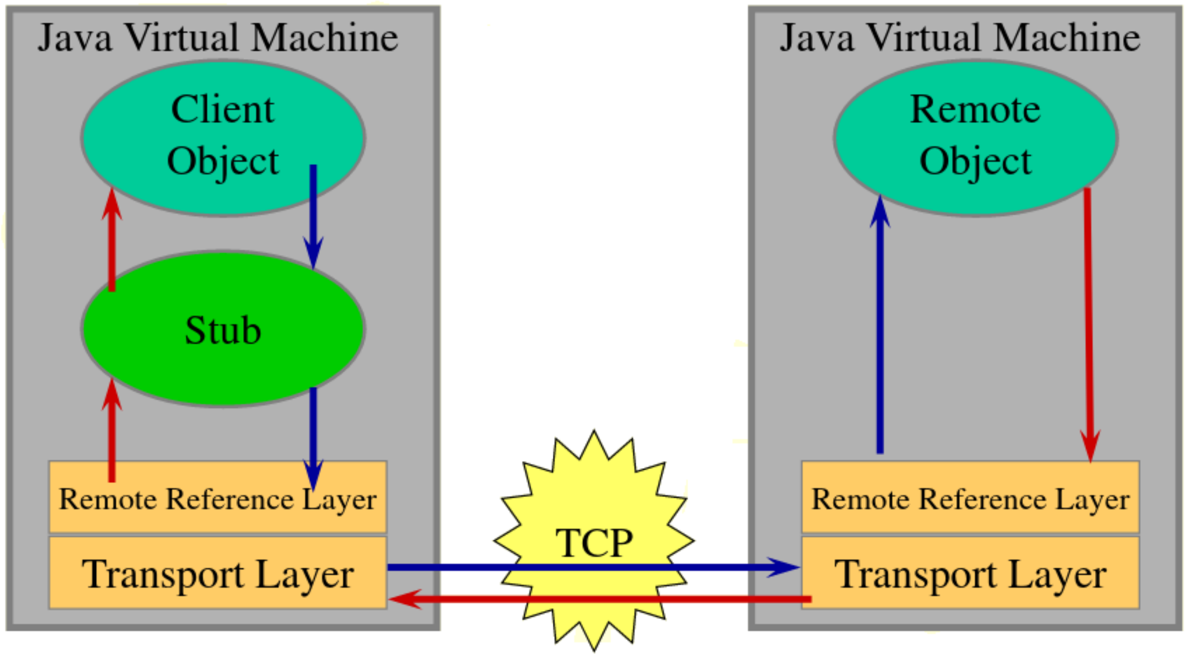


- Objects can be local or remote
- Remote objects are visible through remote interfaces
- RMI masks remote objects as being local using proxy objects
- Object request broker identifies/discovers remote objects

4 Java RMI

- Remote Method Invocation (RMI) is Java's implementation of object-to-object communication among Java objects to realise a distributed system
- RMI allows us to distribute our objects on various machines, and invoke methods on the objects located on remote sites
- **Advantage**: Dynamically invocate new versions of remote objects

- **Application**: Utilize very fast remote processors or any specialised resources



4.1 RMI-based Application Development and Execution

Development:

1. Design the interface for the service
2. Implement the methods specified in the interface

Run time execution:

- On the server
  - Dynamically generate the stub
  - Register the service by name and location
- Client
  - Look up the remote reference on the registry
  - Use the service in an application

4.2 Registries (Object Broker)

A registry is a running process on a host machine

Name and look up remote objects:

- Servers can register their objects
- Clients can find server objects by name and obtain a remote reference
- Clients can obtain a stub for a remote object
- Can also request a list of remote objects from the registry

5 Properties of OOM

Follow object oriented programming model

- Objects, methods, interfaces, encapsulation, (and exceptions)

Synchronous request/reply interaction

- Same as RPC

Location transparency

- Object request broker maps object references to physical locations

Services comprising of multiple servers are easier to built

- All services can be acquired through the object request broker
- Geographical complexity and changes of services are hidden