Neither the client nor the server assume that the procedure call is performed over a network.

**Control flow for RPC calls**

1. **bind to server**
2. **prepare, send request**
3. **unpack reply**

**RPC properties**

**Integration of the RPC into ISO/OSI protocol stack**

- **layer 7 application layer**
  - **client-server model**
  - hides communication details
- **layer 6 presentation layer**
  - message exchange, e.g. request-response protocol
  - Operating system interface to underlying communication protocols
- **layer 5 session layer**
  - transport protocols, e.g. TCP/UDP or OSI TP4
  - transfer of data packets
- **layer 4 transport layer**
  - transport protocols: UDP (User Datagram Protocol) transports data packets without guarantees; TCP (Transmission Control Protocol) verifies correct delivery of data streams.
  - message exchange: socket interface to the underlying communication protocols.
  - RPC: hides communication details behind a procedure call and helps bridge heterogeneous platforms.
There are different types of RPC exchange protocols:
- the request (R) protocol
- the request-reply (RR) protocol
- the request-reply-acknowledge (RRA) protocol.

Integration of software handling the communication between components of a distributed application. Stubs encapsulate the distribution specific aspects. Stubs represent interfaces.

**Client Stub**: contains the proxy definition of the remote procedure P.

**Server Stub**: contains the proxy call for the procedure P.
Client and server stubs have the following tasks during client-server interaction.

1. **Client stub**
   - specification of the remote service operation; assigning the call to the correct server; representation of the parameters in the transmission format.
   - decoding the results and propagating them to the client application.
   - unblocking of the client application.

2. **Server stub**
   - decoding the parameter values; determining the address of the service operation (e.g., a table lookup).
   - invoking the service operation.
   - prepare the result values in the transmission format and propagate them to the client.

How to implement distributed applications based on remote procedure calls?

**Distributed application**

In order to isolate the communication idiosyncrasies of RPCs and to make the network interfaces transparent to the application programmer, so-called **stubs** are introduced.

- **Stubs**
- **Stub functionality**
- **Implementing a distributed application**
- **RPC language**
- **Phases of RPC-based distributed applications**

An **RPC generator**

- reduces the time necessary for implementation and management of the components of a distributed application.
- a declarative interface description is easier to modify and therefore less error-prone.

The individual steps for generating a distributed application are illustrated in the following figure.
The internal structure of a distributed application created using an RPC generator is as follows:

- Client stub (client o)
- Server stub (server o)
- RPC system

- Filters
- Send/receive
- Network
- Generated by RPC generator

Implemented by application programmer.

Manual implementation of stubs is error-prone ⇒ use of a RPC generator to generate stubs from a declarative specification.

**RPC generator**

**Applying the RPC generator**

**Structure of a distributed application**

---

**Distributed applications based on RPC**

How to implement distributed applications based on remote procedure calls?

**Distributed application**

- In order to isolate the communication idiosyncrasies of RPCs and to make the network interfaces transparent to the application programmer, so-called **stubs** are introduced.

  **Stubs**
  
  **Stub functionality**
  
  **Implementing a distributed application**

**RPC language**

**Phases of RPC based distributed applications**

---

**Semistatic binding**

The client determines the server address during the initialization of the client process. Server address remains unchanged for the whole life span of the client process.

**Binding** can take place via:

- Entry in a database.
- Broadcast or multicast message.
- Name service.
- Mediation mechanism ("broker" or "trader"); a broker mediates between client and server.
Possible terms for a mediation component are: registry, broker or trader; Corba uses the term object request broker.

**Functionality of a broker**
- servers register their available service interfaces with the broker ("export interface").
- the broker supplies the client with information in order to locate a suitable server and to determine the correct service interface ("import interface").

**Client-to-server binding**
- broker V
  - 1. export
  - 2. import
  - 3. RPC call
- client C
- server S

**Broker information**

**Handling client requests**
- Broker may either just provide the service interface to the client or act as a mediator between client and server.
  - direct communication between C and S.
  - indirect communication between C and S; communication between C and S is only possible via broker V (or several brokers).

A broker manages information about the available, exported interfaces.
- server names ("white pages")
- service types ("yellow pages")
- behavioral or functional attributes
  - static attributes: functionality of the provided services, cost, required bandwidth.
  - dynamic attributes: current server state.

We distinguish between 3 phases:
  a) design and implementation
  b) binding of components
  c) invocation: a client invoking a server operation.

**Component binding**
**Mediation and brokering**

**Phases of RPC based distributed applications**

**Generated by Torquepos**
**Definition: Remote object** is an object whose method can be called by an object residing on another Java Virtual Machine (JVM), even on another computer.

**Definition: Remote interface** is a Java interface specifying the methods of a remote object.

**Definition: Remote method invocation (RMI)** allows object-to-object communication between different Java Virtual Machines (JVM), i.e., it is the action of invoking a method of a remote interface on a remote object.

The method calls for local and remote objects have the same syntax.

---

**How does RMI work**

Java RMI uses a registry to provide naming services for remote objects, stub and skeleton to facilitate communications between client and server.

1. A server object is registered with the RMI registry
2. A client looks through the RMI registry for the remote object
3. Once the remote object is located, its stub is returned to the client
4. The remote object can be used in the same way as a local object

**RMI architecture**

- **Stub/Skeleton layer**: Layer intercepts method calls by the client and redirects these calls to the remote object.
  - **Remote object**: Object serialization/deserialization; hidden from the application.
  - **Remote Reference layer**: Connects client and remote objects exported by the server environment by a 1-to-1 connection link.
    - **Application layer**: The layer provides JRMP (Java Remote Method Protocol) via TCP/IP.
    - **Presentation layer**: Mapping of stub/skeleton operations to the transport protocol of the host; it interfaces the application code with the network communication.
    - **Session layer**: The layer supports the method `invoke`.
**RMI architecture**

- **Stub/Skeleton layer**
  - Layer intercepts method calls by the client and redirects these calls to the remote object.
  - Object serialization/deserialization; hidden from the application.

- **Remote Reference layer**
  - Connects client and remote objects exported by the server environment by a 1-to-1 connection link.
  - The layer provides J R M P (Java Remote Method Protocol) via TCP/IP.
  - Mapping of stub/skeleton operations to the transport protocol of the host; it interfaces the application code with the network communication.
  - The layer supports the method `invoke`.

  ```java
  Object invoke (Remote obj, java.lang.reflect.Method method, Object [] params, long timeout) throws Exception
  ```

**Locating remote objects**

- **How does the client find the remote object?**
  - RMI supports a special name service, the **RMI registry**
    - Mapping of names to remote objects.
    - Stand-alone Java application.
    - The RMI registry runs on all those machines hosting remote objects.
    - Standard port for registry requests is 1099.
    - The RMI registry is itself a remote object.
  - Access of the RMI registry via the `java.rmi.Naming` class.

**Naming interface methods**

- **Registry Lookup**

```java
public static void bind (String name, Remote obj)
  Throws AlreadyBoundException, java.net.MalformedURLException, RemoteException.
  Associates the remote object with name in URL format.

  Example for name: rmn://host/service-port/service-name
  If name is already bound to an object, then AlreadyBoundException is triggered.

public static void rebind (String name, Remote obj)
  Throws java.net.MalformedURLException, RemoteException.
  Associates always the remote object with name in URL format.

public static Remote lookup (String name)
  Throws NotBoundException, java.net.MalformedURLException, RemoteException.
  Returns a result as a reference (stub) to the remote object.
  If name is not bound to an object, then NotBoundException is triggered.

public static void unbind (String name)
  Throws NotBoundException, RemoteException.

public static String [ ] list (String name)
  Throws java.net.MalformedURLException, RemoteException.
  As a result, it returns all names entered in the registry.
  The name parameter specifies only the host and port information.
```
associates the remote object with name (in URL format).

Example for name: rmi://host/service-port/service-name

if name is already bound to an object, then AlreadyBoundException is triggered.

public static void rebind (String name, Remote obj)

Throws java.net.MalformedURLException, RemoteException.

associates always the remote object with name (in URL format).

public static Remote lookup (String name)

Throws NoBoundException, java.net.MalformedURLException, RemoteException.

returns as a result a reference (a stub) to the remote object.

if name is not bound to an object, then NoBoundException is triggered.

public static void unbind (String name)

Throws NoBoundException, RemoteException.

public static String [] list (string name)

Throws java.net.MalformedURLException, RemoteException.

as a result, it returns all names entered in the registry.

The client invokes a lookup for a particular URL, the name of the service (rmi://host:port/service). The following describes the steps:

1) a socket connection is opened with the host on the specified port.

2) a stub to the remote registry is returned.

3) the method Registry.lookup() on this stub is performed. The method returns a stub for the remote object.

4) the client interacts with the remote object through its stub.