Local vs. remote procedure call

RPC is an extension of the same type of communication to programs running on different computers; single thread of execution and transfer of data.

**Definition**

**RPC properties**

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

**Control flow for RPC calls**

1. **Register service**
2. **RPC-request**
3. **RPC-response**

**Differences between RPC and local procedure call**

For an RPC, the caller and the callee run in different processes.

- **Basic RPC characteristics**
  - Both processes (caller and callee) have no shared address space.
  - No common runtime environment.
  - Different lifetime spans of client and server.

- **Handle errors occurring during a RPC call**, e.g., caused by machine crashes or communication failures.
  - RPC-based applications must take communication failures into consideration.

**RPC vs message exchange**
All RPC can be characterized as follows:

1. uniform call semantics;
2. "type-checking" of parameters and results;
3. parameter functionality;
4. Optimize response times rather than throughput;
5. new error cases

   bind operation failed; request timed out; arguments are too large

   goal is some transparency concerning exception handling and communication failures (relevant for the programmer).

---

Integration of the RPC into ISO/OSI protocol stack:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>application layer</td>
<td>client-server model</td>
</tr>
<tr>
<td>6</td>
<td>presentation layer</td>
<td>RPC</td>
</tr>
<tr>
<td>5</td>
<td>session layer</td>
<td>message exchange; e.g. request-response protocol</td>
</tr>
<tr>
<td>4</td>
<td>transport layer</td>
<td>transport protocols; e.g. TCP/UDP or OSI TP4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Layer</th>
<th>Protocol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>application layer</td>
<td>client-server model</td>
</tr>
<tr>
<td>6</td>
<td>presentation layer</td>
<td>RPC</td>
</tr>
<tr>
<td>5</td>
<td>session layer</td>
<td>message exchange; e.g. request-response protocol</td>
</tr>
<tr>
<td>4</td>
<td>transport layer</td>
<td>transport protocols; e.g. TCP/UDP or OSI TP4</td>
</tr>
</tbody>
</table>

transport protocol: UDP (User Datagram Protocol) transmits 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.

---

RPC vs message exchange:

<table>
<thead>
<tr>
<th>RPC</th>
<th>message exchange</th>
</tr>
</thead>
<tbody>
<tr>
<td>synchronous (generally)</td>
<td>asynchronous</td>
</tr>
<tr>
<td>1 primitive operation (RPC call)</td>
<td>2 primitive operation (send, receive)</td>
</tr>
<tr>
<td>messages are configured by RPC system</td>
<td>message specification by programmer</td>
</tr>
<tr>
<td>one open RPC</td>
<td>several parallel messages possible</td>
</tr>
</tbody>
</table>

The RPC protocol defines only the structure of the request/answer messages; it does not supply a mechanism for secure data transfer.

RPC exchange protocols:

There are different types of RPC exchange protocols:

- the request (R) protocol
- the request-reply (RR) protocol
- the request-reply-acknowledge (RRA) protocol.
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.

**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.

Structure of a distributed application

The internal structure of a distributed application created using an RPC generator is as follows:

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

RPC generator

Applying the RPC generator

Structure of a distributed application