Section 2

Task	Group
The synchronization and asynchronization operations in distributed systems	• دينا إبراهيم جودة . • دينا سامي عيسوي . • خلود أحمد حامد . • خلود عصمت عباس . • دعاء عبدالله قطب .
How to fix the failure handling in client server communication	• خالد أبو المعاطي عبدالسلام . • خالد أحمد عبدالحميد . • رائد إيهاب محمد . • شريف شوقي صالح .
Remote invocation	• ريهام عصام أحمد . • زينب سعيد منير . • سلمى مجدي محمد . • ريهام سعيد فتحي . • سارة محمود محمد .
Request-reply protocol	• رومينا إيهاب مصطفى . • زهراء سعيد بدير . • شروق محمد اللاهوني . • ضحى عبدالمنعم عبدالناصف .
Remote procedure calls	• عبدالحكيم عبدالله عبدالحكيم . • عبدالرحمن السيد عبدالحميد . • عبدالرحمن فتحي عبدالرحمن . • عبدالمنعم سامي إبراهيم . • عبدالوارث جمال عبدالوارث .
Remote method invocation	• علاء حمدي راغب . • عماد جميل انطون . • دينا مجدي الحملاوي . • سمر فتحي الديب . • سارة حسن سليم .

Remote Procedure Call (RPC)

RPC is a high-level model for client-server communication. It provides the programmers with a familiar mechanism for building distributed systems.

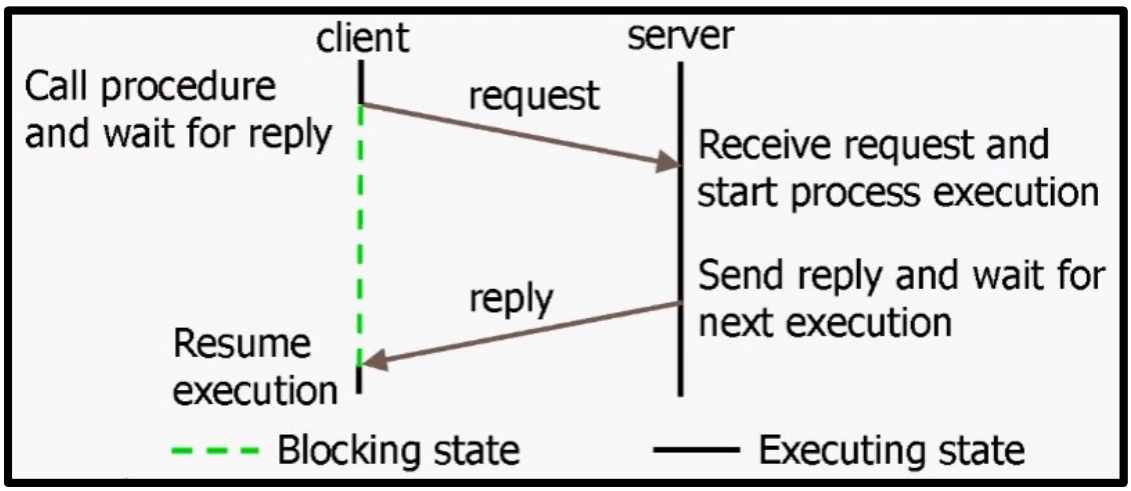
If you have a function *sum(int a, int b)* when you call this function e.g. *sum(5,10)*, the summation isn't done on your computer, but it goes there to the remote computer on which it runs the function(procedure) and then whatever is the result it is given back to your computer.

Why we need RPC?

- The client need an easy way to call the procedure of the server to get some services.
- RPC enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high level languages.
- RPC is modelled on the local procedure call, but the called procedure is executed in a different process and usually a different computer.

How to operate RPC?

- When a process on a machine A calls a procedure on machine B, the calling process on A is suspended, and the



execution of the called procedure takes place on B. As shown in the next figure.

Limitations :-

- Parameters passed by value only and pointer value are not allowed.
- **Speed**: Remote Procedure Calls (and return) time (i.e. overheads) can be significantly (1-3 orders of magnitude) slower than that for local procedure.
    - This may affect real-time design and programmer should be aware of its impact.
- **Failure**: RPC is more vulnerable to failure (since it involves communication system, another machine and another process).
    - The programmer should be aware of the call semantics i.e. programs that make use of RPC must have the capability of handling errors that cannot occur in Local Procedure Calls.

RPC Mechanism

How does the client know the procedure (names) it can call and which parameters it should provide from the server?

- **Server Interface definition**
    - RPC interface specifies those characteristics of the procedures provided by a server that are visible to the clients.
    - The characteristics includes: names of the procedures and the type of parameters.
    - Each parameter is defined as input or output.
- In summary, an interface contains a list of procedures signatures –the names and types of their I/O arguments.

- This interface is made known to the clients through a server process binder.

How does the client transfer its call request (the procedure name) and the arguments to the server via network?

Marshalling and communication with server:-

- For each remote procedure call, a (client) stub procedure is generated and attached to the (client) program.
- Replace the remote procedure call to a (local) call to the stub procedure.
- The (codes in the) stub procedure marshals (the input) arguments and places them into a message together with the procedure identifier (of the remote procedure).
- Use IPC primitive to send the (call request) message to the server and wait the reply (call return) message (DoOperation).

How does the server react the request of the client? From which part? How to select the procedure? How to interpret the arguments?

Dispatching, Unmarshalling, Communication with client:-

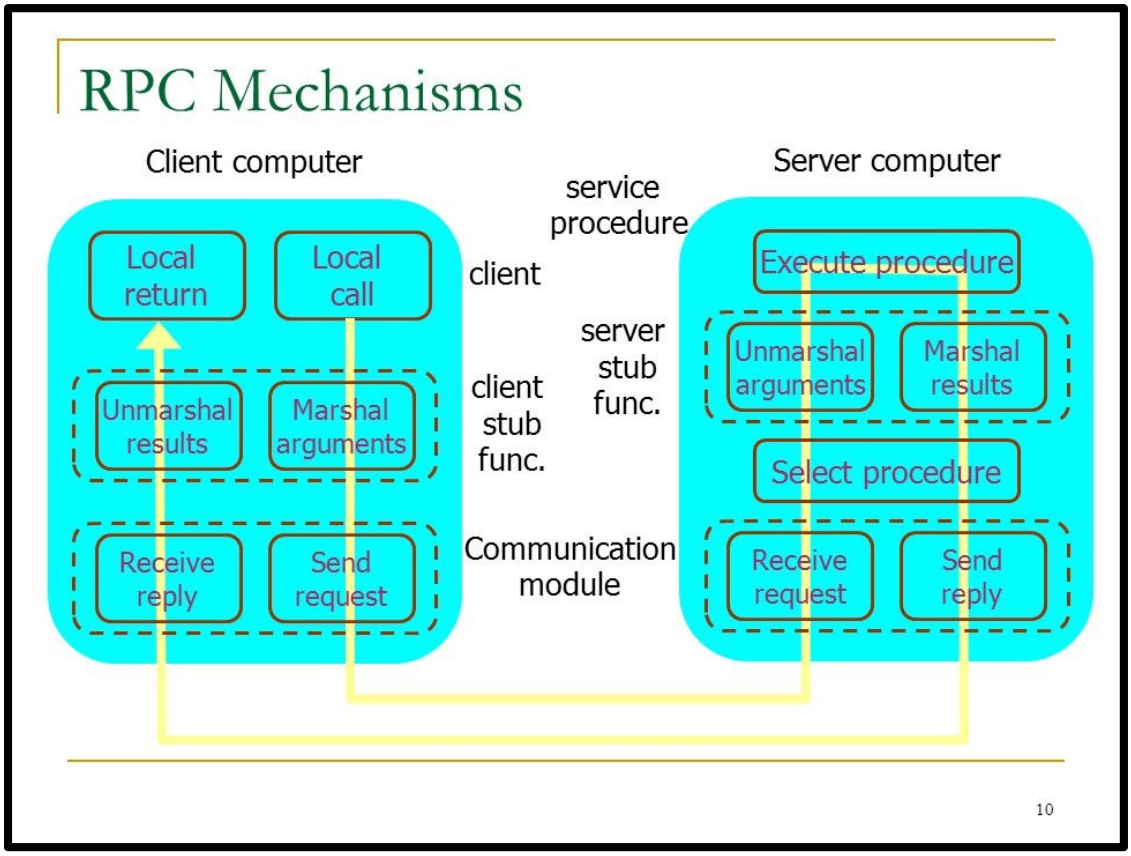
- A dispatcher is provided. It receives the call request message from the client and uses the procedure identifier in the message to select one of the server stub procedures and passes on the arguments.
- For each procedure at the server which is declared (at the server interface) as callable remotely, a (server) stub procedure is generated.
- The task of the server stub procedure is to unmarshal the arguments, call the corresponding (local) service procedure.

How does the server sends the reply back?

- On return, the stub marshals the output arguments into a reply (call return) message and sends it back to the client.

How does the client receive the reply?

- The stub procedure of the client unmarshals the result arguments and returns (local call return). Note that the original remote procedure call was transformed into a (local) call to the stub procedure.



Common issues with QAS Client/Server products and how to resolve them

Problems with TCP/IP Connection

One of the most common configuration issues with QAS Client Server products is in making a TCP/IP connection. This can be caused by a number of factors which will be discussed in more detail below, however there are some basic settings that should be verified before these more complex issues are investigated. These are:

Check that the Server daemon is running. This can be verified in a number of ways:

1. When the server starts, a Process ID number is displayed on the screen.

Use the command **ps** to display process statistics. This command is implemented as follows:

ps –f –l –u<username>

This will list all the users processes and some of the attributes of these process as shown below:

F S USER PID PPID C PRI NI ADDR SZ WCHAN STIME TTY TIME CMD

40801 S romanur 8858 1 2 61 20 895 1896 1fd67c 17:56:39 pts/1 0:00 ./qanstdp

241801 S romanur 9608 7559 0 60 20 7a9 120 17:52:52 pts/1 0:00 -ksh

200001 R romanur 9883 9608 5 62 20 5ce 132 17:56:46 pts/1 0:00 ps -f -l

The statistics above show the state of the process **qanstdp** and confirms that the process is running and clocking up processor time. The number 2 under the "C" column indicates the CPU utilisation. The larger the number the more CPU intensive is the process.

2. Ensure that the normal connection between the client and the server is sound. This can be achieved by running other software which is common to both the client and the server. Alternatively the **PING** command can be used to try connecting to the HOST, where the server resides.

The syntax for using PING is as follows :

ping <hostname>

where hostname is the name of the server you want to connect to.

You should get a similar response to below for a successful connection using **PING** :

Pinging 150.150.100.240 with 32 bytes of data:

Reply from 150.150.100.240: bytes=32 time=2ms TTL=64

Reply from 150.150.100.240: bytes=32 time=1ms TTL=64

Reply from 150.150.100.240: bytes=32 time=1ms TTL=64

Reply from 150.150.100.240: bytes=32 time=1ms TTL=64

If PING fails then the problem is with the System setting. Possible problems here are that network protocols such as TCP/IP have not been properly installed or there are real network problems between the server and the client. In this instance it may be necessary to have a network analysts trouble-shoot the network.

If PING proves to be successful, it indicates that the network is sound.

3. The next area to examine would be to double check that the Dotted IP numbers in the INI file setting "**ServerAddress=**" are correct and that the Port number is the same for the server and client. If this fails it may be worth trying to change the port number on the server and the client and then re-starting the QuickAddress Network Server.

Another diagnostic test that one can perform is to verify that the server is running correctly by running telnet. This is done by specifying the port number that the server is running on in the port number text box. If you get **"QAS1"** appearing on the screen, termed as a handshake, the port number is valid and the server is listening. If the handshake fails then it indicates that there are problems and the server is not listening, consequently there will be no connection.

Some servers may have multiple network cards, thus they will have individual IP addresses and this could sometimes cause a problem. If multiple network cards exist then the **ServerAddress** setting in the INI file should be edited to read

ServerAddress=Any

If all the above possible solution fixes fail then kernel logging facility can be implemented by including the following lines in the relevant product INI file:

LogTrace=1

LogFile=<filename>

LogErrors=True

4. Another utility which can be used in troubleshooting such connection problems is **netstat.** This indicates whether a connection is established or not with a particular port number. This is implemented in the command line of Windows /UNIX environments as follows:

netstat –a

Commonly Encountered QAS TCP/IP Error Messages

There are two types of errors which you might encounter when QAS Client/Server Products are running. These are:

1. Errors generated by the Names Server and sent to the Names Client.

2. Errors arising in the network communication between the Names Server and Names Client, thus stopping these programs from initialising or communicating.

(Under Windows NT, the error values returned are positive values, rather than negative)

Error -9291 ("Bad IP number format")

This error occurs if the IP number specified when installing the Network Client is incorrectly formatted. This number should be in 'dotted quad' format - i.e. four numbers separated by dots (for example: 150.150.100.50).

Solution: Correct the formatting in QADDRESS.INI

Amend the ServerAddress= setting in the Client's configuration file to the correct IP number. Alternatively, you can specify a hostname in this setting instead of an IP address.

Error -9293 ("Couldn't listen to TCP/IP socket")

This error probably means that the port you have specified is already in use by another application.

Solution: Check the ServerPort setting

The ServerPort= setting in the Client's configuration file contains the port number which the Server is using to receive requests. Check that this setting corresponds with the ServerPort= setting in the Server's configuration file.

Error -9294 ("Couldn't bind TCP/IP socket (port in use?)")

This error could occur when the Server is started. The system believes that the port specified by the ServerPort configuration setting is unavailable or is already in use.

Such an error may be encountered if a copy of the Server is already running, or if a different program running on the Server

machine has already claimed the port that the Names Server is looking for.

Error -9292 ("TCP/IP accept failure")

This error will occur if the connection between Client and Server has not been made. This might be due to a network problem.

Solution: Check the Server is running

Ensure that the QAS Network Server is loaded and running before trying to connect to it. Check that it hasn't crashed and that there are no broken network connections.

Check the Client configuration file

If the Server is running, then ensure that the ServerAddress= setting in the Client configuration file is correct, and that the ServerPort= setting corresponds with the equivalent setting in the Server configuration file.

Error -9296 ("TCP/IP timeout")

This error may occur if data transfer between Server and Client is not completed within the connection timeout period. This might be due to a broken connection or a high volume of network traffic.

Solution: Provide more specific search address information

Large amounts of data are transmitted if search results have many pick list items. If you specify more name and address information, the search will be narrower, meaning less data to return.

Error -9297 ("TCP/IP socket error")

This error appears when the Client has difficulty sending to a port, possibly because the Server has crashed or because the user does not have authorisation to use the specified port number.

Solution: Check whether the required port is available

Do you run the Server under UNIX? If so, then unless you are running the Server from an account with root privileges you must use port number **1024** or greater.

If you are a UNIX user, you can investigate "/etc/services" to check if any other service is using the port you've configured. Alternatively, you can use the **netstat** utility discussed above.

On NT, you can check the SERVICES file or also use **netstat**.

If the Server is definitely running, check that it hasn't crashed. Check that there are no broken network connections. Most importantly, ensure that the Server's name and port settings in the configuration file have been correctly specified.

Error -9298 ("Couldn't open TCP/IP connection")

This error occurs when starting a Client which cannot establish contact with the Server, either because the Server is not running or because the Server address and port number have been incorrectly specified.

Solution: Check the status of the server

- If the Server is **NOT** running:

To prevent this error, you must ensure that the Server is loaded and running before any Client attempts to connect to it.

- If the Server **IS** running:

Check that the Server hasn't crashed. Check that there are no broken network connections. Most importantly, ensure that the Server's name and port settings in the configuration file have been correctly specified.

Error -9299 ("Couldn't create TCP/IP socket")

This error occurs if the TCP/IP port specified is already in use or the user doesn’t have permission to use it.

Solution: Check if the requested port is available

If the Server is running under UNIX, then unless it is running from an account with root privileges, a port number of **1024** or greater must be used.

If you are a UNIX user, you can investigate "/etc/services" to check if any other service is using the port you've configured. Alternatively, you can use the **netstat** utility discussed above.

Error -9300 ("Dataset not present on Server")

This error occurs if a DataPlus data set specified in the Client configuration file is not in the Server configuration file.

Solution: Compare Client and Server configuration files

Make sure that the DataPlus data sets specified in the Client's configuration file are also in the Server's configuration file. The Client does not have to use every DataPlus set that is configured on the Server, but a data set must be on the Server before a Client can retrieve information from it.

Error -9301 ("Incorrect Client/Server protocol (old version?)")

This error might occur if the Client and Server have different release numbers (i.e. one has been upgraded and the other has not). In such circumstances it is necessary to contact Experian QAS Technical Support.

Synchronous and Asynchronous in Distributed Systems

The way we reason about properties of a distributed system or attempt to develop a mental model or abstraction for the system more or less depends on the nature of distributed system — whether synchronous or asynchronous.

Synchronous Distributed System

A synchronous distributed system comes with strong guarantees about properties and nature of the system. Because the system makes strong guarantees, it usually comes with strong assumptions and certain constraints. Synchronous nature by itself is multi-faceted, and the following points will elaborate more on this:

1- **Upper Bound on Message Delivery** There is a known upper bound on message transmission delay from one process to another process OR one machine/node to another machine/node. Messages are not expected to be delayed for arbitrary time periods between any given set of participating nodes.

2- **Ordered Message Delivery** The communication channels between two machines are expected to deliver the messages in FIFO order. It means that the network will never deliver messages in an arbitrary or random order that can't be predicted by the participating processes.

3- **Notion of Globally Synchronized Clocks** Each node has a local clock, and the clocks of all nodes are always in sync with each other. This makes it trivial to establish a global real time ordering of events not only on a particular node, but also across the nodes.

4- **Lock Step Based Execution** The participating processes execute in lock-step. An example will make it more clear. Consider a distributed system having a coordinator node

that dispatches a message to other follower nodes, and each follower node is expected to process the message once the message is received. It cannot be the case that different follower nodes process the input message independently at different times and thus generate output state at different times. This is why we say processes execute in lock step synchrony a la lock step marching. The main thing to remember about synchronous systems is that they allow us to make assumptions about time and order of events in a distributed system. This comes from the fact that clocks are in sync and there is a hard upper bound on message transmission delay between nodes.

The problem with synchronous distributed systems is that they are not really practical. Any software system based on strong assumptions tends to be less robust in real world settings and begins to break in practical/common workloads

Asynchronous Distributed System

The most important thing about an asynchronous distributed system is that it is more suitable for real world scenarios since it does not make any strong assumptions about time and order of events in a distributed system.

1-Clock may not be accurate, clocks can be out of sync Clocks of different nodes in a distributed system can drift apart. Thus it is not at all trivial to reason about the global real time ordering of events across all the machines in the cluster. Machine local timestamps will no longer help here since the clocks are no longer assumed to be always in sync.
2-Messages can be delayed for arbitrary period of times Unlike synchronous distributed system, there is no known upper limit on message transmission delay between nodes. Asynchronous distributed system is tough to understand since it is not based on strong assumptions and does not really impose any constraints on time and ordering of events. It is

also tough to design and implement such a system since the algorithms should tolerate different kinds of failures.

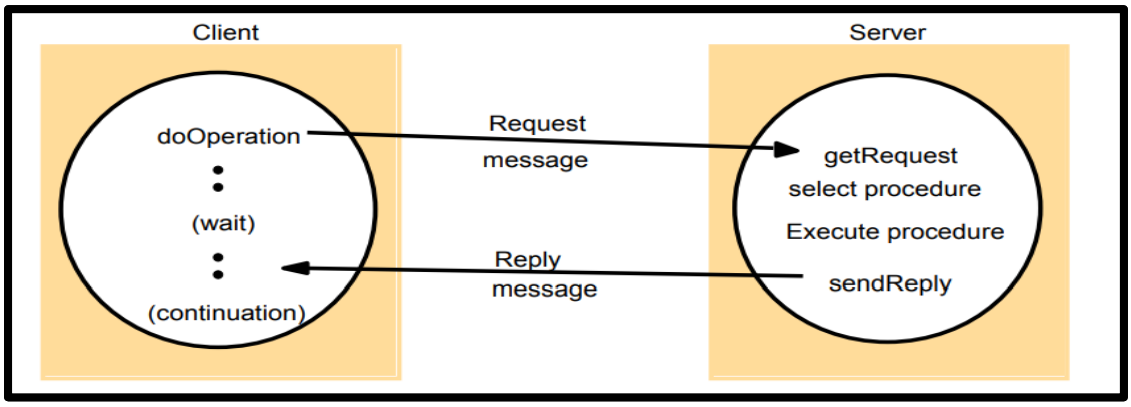
Our algorithms can no longer be designed to handle only a subset of failure conditions by ruling out some failure scenarios using strong assumptions. The onus and challenge of developing robust distributed algorithms is more in asynchronous distributed system.

Request/Reply Protocols

Request/Reply Communication method

Is one of the basic techniques computers use to communicate with each other and to request the services from one another.

In this method the first computer sends a request for some data and the second computer responds to the request. Usually, there is a series of such interchanges until the complete message is sent; browsing a web page is an example of request–reply communication. Request–reply can be seen as a telephone call, in which someone is called and another one answer the call.



Request–reply is a message exchange pattern in which a requestor sends a request message to a replier system which receives and processes the request, ultimately returning a message in response. This is a simple, but powerful messaging pattern which allows two applications to have a two-way conversation with one another over a channel.

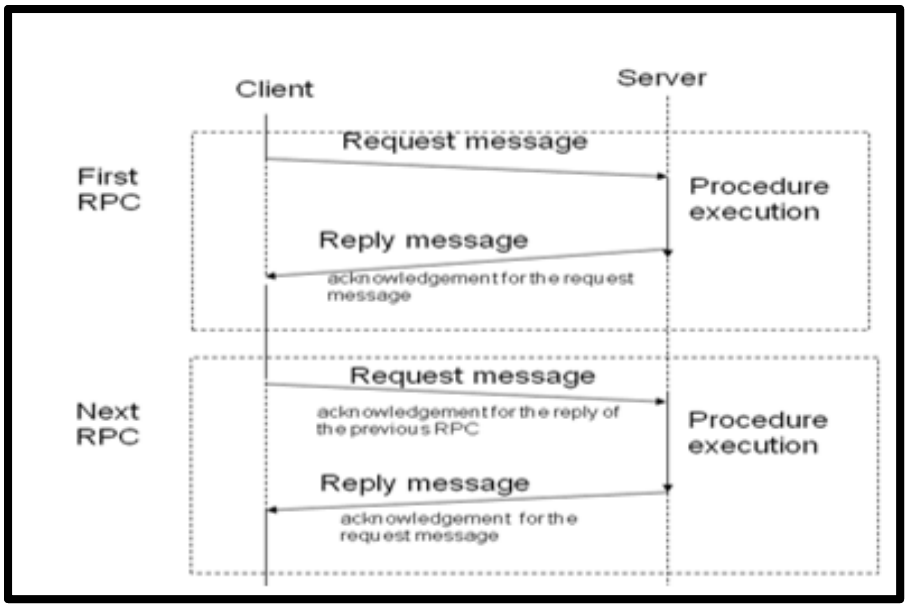
This pattern is especially common in client–server architectures.

This pattern is typically implemented in
purely synchronous fashion, as in web service calls over HTTP, which holds a connection open and waits until the response is delivered or the timeout period expires. However, request–reply may also be implemented asynchronously, with a response being returned at some unknown later time. This is often referred to as "sync over async", or "sync/async", and is common in enterprise application integration (EAI) implementations where slow aggregations, time-intensive

functions, or human workflow must be performed before a response can be constructed and delivered.

There are two types of Request/Reply protocols

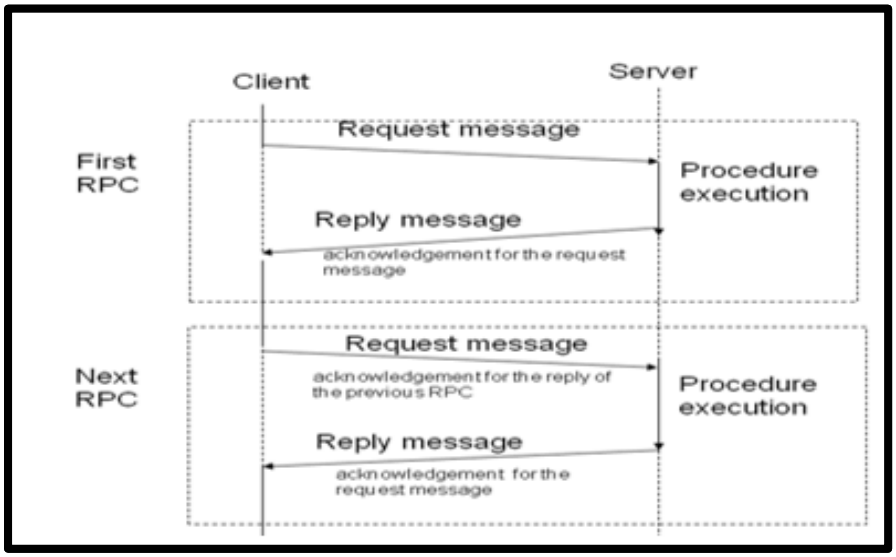
1-The Request/Reply protocol (implicit ACK)



- This protocol is also known as RR(request/reply) protocol.
- It is useful for designing systems which involve simple RPCs.
- In a simple RPC all the arguments and result fit in a single packet buffer while the call duration and intervals between calls are short.
- This protocol is based on the idea of using implicit acknowledgement to eliminate explicit acknowledgement messages.
- In this protocol a server reply is considered as an ACK for a clients request and a subsequent call from a client is considered as ACK of the client's previous call.
- Timeout-and-retires technique is used with RR protocol for failure handling. Retransmission of request message is done when there is no response.
- RR protocol and timeout technique provides at-least-once call semantics on if duplicate requests are not filtered out.

- Exactly once semantics are supported by servers using reply cache which stores replies.

2-The Request/Reply/Acknowledgement-Reply Protocol



- This protocol is also known as RRA (request/reply/acknowledge-reply) protocol.
- RR protocol implements exactly once semantics which requires storage of a lot of information in the server cache and can lead to loss of replies that have not been delivered.
- To overcome the limitations or RR protocol, RSA protocol is used.
- In this clients acknowledge the receipt of reply messages and the server deletes information from its cache only after it receives an acknowledgement from client.
- Sometimes the reply acknowledgement message may get lost therefore RRA protocol needs a unique ordered message identifiers. This keeps a track of the acknowledgement series sent.

Remote Method Invocation (RMI)

Definition :

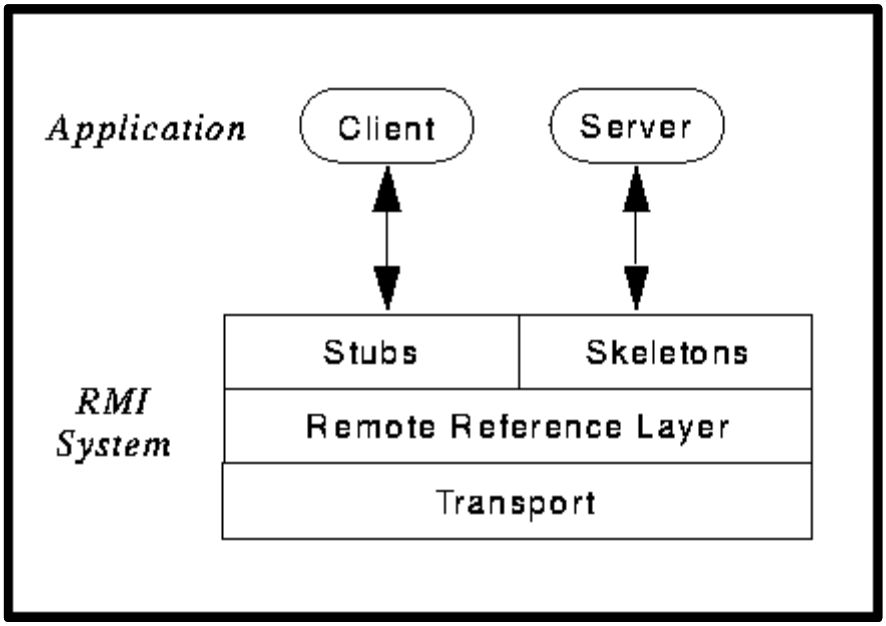
is a way that a programmer, using the Java programming language and development environment, can write object-oriented programming in which objects on different computers can interact in a distributed network.

Structure :

The RMI system consists of three layers:

- The stub/skeleton layer - client-side stubs (proxies) and server-side skeletons .
- The remote reference layer - remote reference behavior (such as invocation to a single object or to a replicated object) .
- The transport layer - connection set up and management and remote object tracking .

The boundary at each layer is defined by a specific interface and protocol; each layer, therefore, is independent of the next and can be replaced by an alternate implementation without affecting the other layers in the system.



Advantages :

- Object Oriented: RMI can pass full objects as arguments and return values, not just predefined data types. This means that you can pass complex types, such as a standard Java hashtable object, as a single argument.
- Mobile Behavior: RMI can move behavior (class implementations) from client to server and server to client. For example, you can define an interface for examining employee expense reports to see whether they conform to current company policy. When an expense report is created, an object that implements that interface can be fetched by the client from the server.
- Design Patterns: When you can pass behavior, you can use object oriented design patterns in your solutions. All object oriented design patterns rely upon different behaviors for their power; without passing complete objects-both implementations and type.
- Safe and Secure: RMI uses built-in Java security mechanisms that allow your system to be safe when users downloading implementations. RMI uses the security manager defined to protect systems from hostile applets to protect your systems and network from potentially hostile downloaded code.
- Easy to Write/Easy to Use: RMI makes it simple to write remote Java servers and Java clients that access those servers. A remote interface is an actual Java interface.
- Connects to Existing/Legacy Systems: RMI interacts with existing systems through Java's native method interface JNI. Using RMI and JNI you can write your client in Java and use your existing server implementation.
- Write Once, Run Anywhere: RMI is part of Java's "Write Once, Run Anywhere" approach. Any RMI based system is 100% portable to any Java Virtual Machine

- Distributed Garbage Collection: RMI uses its distributed garbage collection feature to collect remote server objects that are no longer referenced by any clients in the network.
- Parallel Computing: RMI is multi-threaded, allowing your servers to exploit Java threads for better concurrent processing of client requests.
- The Java Distributed Computing Solution: RMI is part of the core Java platform starting with JDK?? 1.1, so it exists on every 1.1 Java Virtual Machine.

How it work :

A remote method invocation from a client to a remote server object travels down through the layers of the RMI system to the client-side transport, then up through the server-side transport to the server. A client invoking a method on a remote server object actually makes use of a *stub* or proxy for the remote object as a conduit to the remote object. A client- held reference to a remote object is a reference to a local stub. This stub is an implementation of the remote interfaces of the remote object and forwards invocation requests to that server object via the remote reference layer.The *remote reference layer* is responsible for carrying out the semantics of the invocation. For example, the remote reference layer is responsible for determining whether the server is a single object or is a replicated object requiring communications with multiple locations. Each remote object implementation chooses its own remote reference semantics-whether the server is a single object or is a replicated object requiring communications with its replicas. Also handled by the remote reference layer are the reference semantics for the server. The remote reference layer, for example, abstracts the different ways of referring to objects that are implemented in (a) servers that are always running on some machine, and (b) servers that are run only when some method invocation is made on them (activation). At the layers above the remote reference layer, these differences are not seen. The *transport layer* is

responsible for connection setup, connection management, and keeping track of and dispatching to remote objects (the targets of remote calls) residing in the transport's address space.

In order to dispatch to a remote object, the transport forwards the remote call up to the remote reference layer. The remote reference layer handles any server-side behavior that needs to occur before handing off the request to the server-side skeleton. The skeleton for a remote object makes an up call to the remote object implementation which carries out the actual method call.

The return value of a call is sent back through the skeleton, remote reference layer, and transport on the server side, and then up through the transport, remote reference layer, and stub on the client side.

Remote Invocation

. Deals with some enhancements of PIC.

. Includes invoking remote elements.

. Methods of PIC.

- Request reply protocol(RRP): Requestor requests with a message and gets a reply.
- Remote produce call(RPC): Extension of normal function calling where the called and calling procedure are not in the same address space .
- Remote Method Invocation (RMI): Use of OOP concept in distributed event.