Global Scientific JOURNALS

GSJ: Volume 9, Issue 4, April 2021, Online: ISSN 2320-9186
www.globalscientificjournal.com

Message Passing: Survey on RPC, RMI, and CORBA

Abdulrahman Osama Mustafa
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia
ahelmymostafa@stu.kau.edu.sa

Abdulmajeed Abdullah Abbas
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia
aaabbas@kau.edu.sa

Muhamad Alamin Sayegh
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia
mmohamadsayegh@stu.kau.edu.sa

Prof. Mohammed Jaffer Alhaddad
Faculty of Computing and Information Technology
King Abdulaziz University
Jeddah, Saudi Arabia
malhaddad@kau.edu.sa

***Abstract*—Message passing is a critical part of any distributed system. It allows the different components of a distributed system to communicate with each other and allow clients to use the services it provides, send commands, and receive results. Many technologies implement message passing such as Remote Procedure Call (RPC), Remote Method Invocation (RMI), and Common Object Request Broker Architecture (CORBA). This paper presents an overview of these technologies and a survey on publications that are available on them.**

***Index Terms*—Message Passing, Remote Procedure Call, RPC, Remote Method Invocation, RMI, Common Object Request Broker Architecture, CORBA.**

I. Introduction

In distributed computing, a protocol for sending requests and receiving a response is very important and it is often referred to as message passing. When running an application on a local computer, this application will call functions and procedures and pass parameters around very easily using memory by either copying the data from the caller's memory to the callee's memory or by copying the address of the caller's memory. The callee, then, will operate on this data and return results.

This is no longer possible in a distributed system because each component is on a different computer and there is no shared memory between them. The only way to communicate with each other is by using a network protocol and request-response-based communication.

There are many ways of implementing message passing. One way is to extend the concept of calling functions and procedures into the distributed world and this is where the concept of Remote Procedure Call (RPC) comes from. It isa protocol for message passing where clients invoke methods on another computer as if they were on the same computer.

With the popularity of Object-Oriented Programming(OOP), Oracle decided to design an RPC system that is based around objects so they developed the Remote Method Invocation (RMI) using Java language. RMI allows a client to invoke remote object implementation using the network to the server and invoke member methods on that object. RMI will take care of serializing the object and sending the object byte code to the server to run the code dynamically.

RMI is a Java-specific system it cannot run on other programming languages and most RPC implementations are also platform-dependent or OS-dependent. Object Management Group (OMG) decided to design a standard for communication between systems that are different using the distributed object paradigm called Common Object Request Broker Architecture (CORBA). CORBA is standard designed to provide interoperability among distributed objects independent from the hardware, operating system, and programming language. One object written in C++ for example can communicate with an object written in java because they both use the CORBA standard. One of the components in CORBA that allow it to be language-independent is the Interface Definition Language (IDL) which is used to define the interface of the object without having to write it in a specific language. The IDL is then converted by the CORBA implementation to the target language for use.

In this paper, we present an overview of RPC, RMI,

and CORBA from various perspectives, we analyze multiple implementations of RPC and the difference between them, andwe show some of the features and characteristics of RMI beingan implementation of RPC. We also show an

Overview of CORBA and its components and the role of each one, we review some of the publications on fault-tolerant CORBA and how it is achieved, and we analyze the Concurrency Control Service standard of CORBA that is usedfor synchronization.

II. REMOTE PROCEDURE CALL (RPC)

RPC (Remote Procedure Call) is a simple and commonly used paradigm for building distributed applications. The Net-work Computing Architecture (NCA) [8] is an example of a distributed system that uses it as a means of communication. Despite the fact that RPC is a clear and straightforward definition, there are many subtle and challenging problems [9]. In the design of various RPCs, some device parameters have to be traded off against other parameters. As a result, there are various RPC implementations available in both the research and industrial settings.

The main aim of this section is to compare and contrast a few different RPC implementations, including their design focus, strategies, strengths, and weaknesses.

A. RPC design

The RPC work in two processes:

1) The process of doing the call (the invoker or client).
2) The process created to service the call (the server).

The client-server paradigm is used in RPC. The client is the program that makes the order, and the server is the program that provides service. An RPC is a synchronous process that requires requesting software to paused before results of the remote procedure were returned, similar to a normal or a local procedure call. Many RPCs can be executed simultaneously by using lightweight processes or threads that occupy the same address space [28] [29].

These processes are usually contained in various objects, and they may even be located on diverse virtual or physical machines. The invoking method waits for the call's results to be returned. As a result, from the client's viewpoint, a remote procedure call is synchronous as shown in Fig. 1.

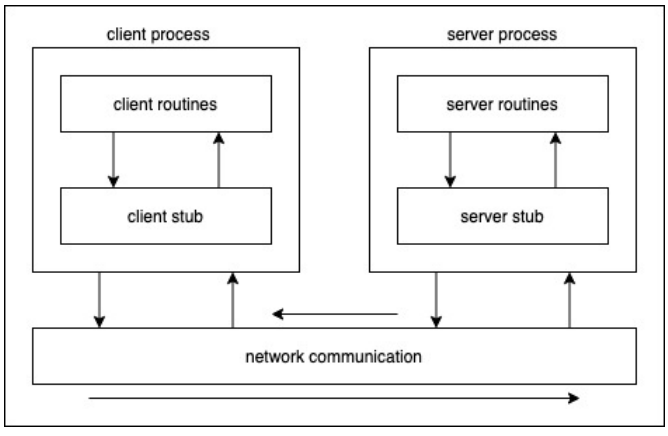

Fig. 1: RPC design and integration

B. Interface definition language (IDL)

Remote Procedure Call software often employs the interface description language, is a specification language for describinga software components application programming interface (API). In this case, IDL acts as a connection between machinesat each end of the link, which could be running various operating systems and programming languages.

C. RPC message procedure

When function statements that use the RPC specification are compiled into a runnable program, a stub that represents remote procedure code is included in compiled code. The stub collects requests and directs them into the client runtime program in a local machine until the program is executed and the procedure call is released. When the database stub is called for the first time, It interacts with a name server to decide the server's transport address [28] [30].

The client runtime program knows how to handle the remote machine and server application and transfers the order for the remote process through the network. A runtime software and stub that communicate with the remote procedure are also included on the server. The same is true for response-request protocols.

D. RPC Synchronous/Asynchronous

RPC operations such as sending, receiving, and replying can be synchronous or asynchronous, or a combination of both. A synchronous operation prevents process from continuing until operation is completed. Asynchronous operations do not block and only start the process. [10]

Understanding what it means for an operation to complete required for synchronous operations. When a message is delivered to receiver via remote assignment, both the send and receive processes are completed. If there is a return value, the send, receive, and reply completely when the result is delivered to the sender in the case of a remote procedure call. In any case, when the procedure is finished, the send and receive are complete. As previously stated, the sender and receiver are in a rendezvous during the procedure's execution. [11] [12]

E. RPC Performance

Many distributed systems rely on the RPC for communi-cation. As such, the component's performance is crucial. As a result, a great deal of research has been done to improve the RPC implementations. Many studies have been performed. Declarative, uninteresting arguments, phrases, error handling, casts and stage of a function call are omitted from code listings for clarification. they usually result in the use of modern protocols that are incompatible with existing specifications like the Sun RPC. [32].

F. RPC Optimization

RPC has gone through a lot of research and improvements A direct derivation of an improved version from existing code is an alternative to re-implementing a device feature for performance reasons. Starting with existing code has the advantage of having the derived version consistent with existing standards. The systematic derivation method can also

be replicated for various machines and systems, which is an added benefit.

components, in reality, are known to be generic and organized in layers and modules.

This results in different ways of interpretation, which are significant sources of overhead. for example, HP-UX file systems [31], and optimization in Sun RPC by several layers of functions that interpret descriptors to decide communication parameters: Protocol (TCP or UDP), encoding or decoding, and buffer management are all options [32].

G. RPC Latency

When RPC is used to link processes over a wide-area network, the protocol must support location services as well as direct communication between the processes. Since wide-area networks have such a high latency rate, an acceptable protocol for local-area networks will be ineffective.

To facilitate the interconnection of local-area networks, the Amoeba distributed operating system [33] added a session layer gateway. Target servers export their port and wide-area network address to other Amoeba sites using the publish feature. Each site installs a server agent after receiving this information.

For wide-area communication, it uses whatever protocol is available and without the client and server processes knowing about it. For local communication, it uses protocols optimized for local networks. The error recovery is very powerful in this model because the client agent notifies the server agent and the reverse when a shut-down of the client occurs.

H. RPC Security

The provision of data privacy and authentication in such an open communication network is a big challenge introduced by an external communication network. There are some concerns about security in an RPC mechanism [35]:

- **Authentication:** To verify the identity of each caller.
- **Availability:** To ensure that callee access cannot be maliciously interrupted.
- **Secrecy:** To ensure that callee information n is disclosed only to authorized callers.
- **Integrity:** To ensure that callee information is not destroyed.

Systems address these topics in various ways, giving more insight or prioritizing them depending on their most relevant implementations. The Cedar RPC Facility [34] Distributed Database serves as a data encryption authentication tool or key delivery center.

Protection is supported by Andrew's RPC process [36]. A connect procedure is used when a caller needs to coordinate with a callee. The linking creates a conceptual relation at one of the system's four levels:

- **OpenKimono:** the information is neither authenticated nor encrypted.
- **AuthOnly:** the information is authenticated, but not encrypted.

The obvious question at this point is whether there are substantial opportunities to derive dramatically improved versions of existing system components. Many current system

- **HeadersOnly:** the information is authenticated, and the RPC packet headers, but not bodies, are encrypted.
- **Secure:** the information is authenticated, and each RPC packet is fully encrypted.

III. REMOTE METHOD INVOCATION (RMI)

As defined by [1], Java Remote Method Invocation (RMI) is a sort of mechanism that allows a Java virtual machine (JVM) to invoke methods located in a remote server or another JVM by calling its object method. RMI is known as a fundamental concept in the world of distributed systems. [2] claimed that RMI allows distributed objects to easily implemented using its architecture that based on two essential separated programs:

A. RMI Design

- **Server:** creates various remote objects assigned to references and makes them accessible by these remote object references, then waits for clients calls to methods on these remote objects, (see Fig. 2).
- **Client:** gets remote references to remote objects in the server and invokes their methods.
- **Stub:** is an image of the remote object at the client side. It operates as an entryway.
- **Transport Layer:** this layer is the link between the server and the client for the current and new connections.
- **Skeleton:** the object which exist in in server side. Client-side stub communicates with this skeleton in order to deliver request to the remote object.
- **Remote Reference Layer (RRL):** a managing layer that panels references made by client to the remote object.

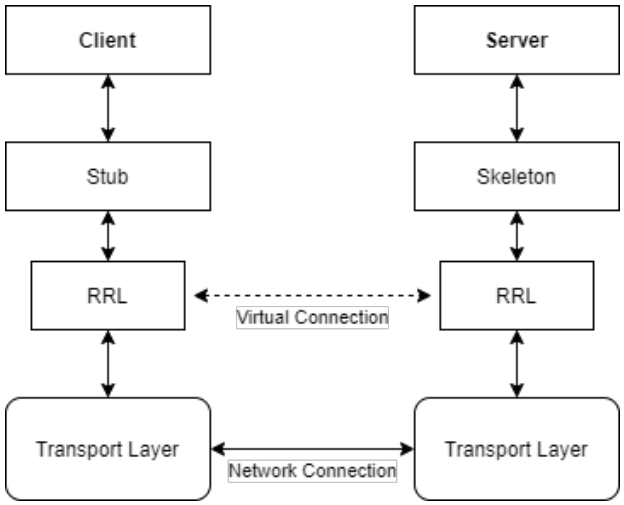

Fig. 2: RMI Architecture

B. RMI Advantage

While RMI is one of three key standards of distributed object technology besides DCOM (Distributed Component Object Model), and CORBA (Common Object Request Broker Architecture), as it surpasses the other two by defeating some limitations in platform and realization complexity. Because

RMI is a pure Java distributed solution, JVM objects can communicate to each other on different machines and memories, and via other physical devices.

C. RMI Optimization

For the purposes of optimizing the performance level of this architecture over TCP in procedural communication mechanism, a study [3] has been made of how this architecture works and how it behaves in different programming models. The result showed that the possibility of converting RMI into a form that allows it to be used in the client's side asynchronously with an obvious increase in performance.

D. RMI Latency

One of the issues of RMI in wide environment use is the high latency which can be noticed in the performance of Java applications. The most-common simple solution for this issue is cashing objects at the client-side, which, in turn, could lead to further issues such as distortion of consistency. From this point, [4] proposed two technicians for managing the consistency of the objects cashing in RMI-based applications, thus, the system designer can choose the proper strategy for the application from these two mechanisms:

1) *Time Stamp technique (TS-RMI)*
   In this technique, Time comparison is done between modified times in a server with cashed time on the client-side.
2) *Invalid message technique (IM-RMI)*
   In this technique, whenever the object changed in the server, it broadcasts object updating messages to all clients that used that cached object.

After some experiments, the results show whenever the frequency of the server is high, TS-RMI is faster than IM-RMI in response time. Otherwise, IM-RMI is faster than TS-RMI in response time.

E. Optimizing RMI in Clusters

Using Java for parallel programming on clusters is limited by weak support of high-speed in clusters and the lack of efficient communication middleware that delayed its operation. [5] presents a way for implementing Java RMI in clusters in a more efficient way to overcome this limitation without any source code modifications, totally transparent to the user, and compatible with other systems. While performance plays an essential role in parallel computing, there were some attempts to develop an effective middleware for Java distributed shared memory e.g CoJVM [6]. The main goal of [3] is to deliver high-performance and large support for Java RMI implementation. This is done by using some specific sockets library that handles the requirements of RMI in parallel computing and by optimizing RMI protocol under some essential assumptions for the targeted used mechanism. The optimization focus on three aspects:

a) Transport Protocol Optimization
b) Serialization Overhead reduction
c) Object Manipulation Improvements

And the results show that the overhead of calls is clearly reduced and hence improves performance.

Furthermore, [7] report their work for providing a high-performance RMI mechanism used in Common Component Architecture (CCA), which allows CCA applications to astoundingly use parallel systems to speed up calculations operations. Their work relies on the previous Babel tool which is a way that enables interoperability of different languages codes to invoke each other.

The paper tries to deliver a high-performance RMI protocol by exploiting three main features:

- high-performance feature via introducing little latency as possible.
- portability feature by executing on common high-performance computing platforms.
- Lightweight by trying to not trouble the CPU utilization by the computing operation.

These features, successfully benefit all the scientific software in CCA.

F. RMI Security

Java RMI security level considered as very low especially for production systems, [14] used different technologies that cover two of the three fundamental principles of information security; integrity, confidentiality, and authentication by using Kerberos and Java Authentication and Authorization services (JAAS) to enhance the security level of RMI and build a Secure RMI library as a result.

G. Synchronization

Java supports the creation and monitoring synchronization of threads by waiting and synchronize any object, but it does not work the same way for remote objects. And the synchronization methods do not work in Java RMI. [27] presented a mechanism that adds the thread synchronization to Java in distributed systems. Their technique for monitor-style has been applied in the context of J-Orchestra, which is a system that rewrites current Java classes at the bytecode into distributed programs that can execute on various machines. The technique solves the absence of matching between the Java concurrency mechanism and the middleware.

IV. COMMON OBJECT REQUEST BROKER ARCHITECTURE (CORBA)

Common Object Request Broker Architecture is a specification framework that was developed by Object Management Group (OMG) to unify computing across different hardware, operating systems, and languages by providing a message passing mechanism. OMG is an international computer standards consortium that develops enterprise integration standards. CORBA is their attempt at moving the object-oriented (OO) programming paradigm into distributed computing [23].

A. CORBA Components

CORBA consists of 5 main components as shown in Fig 3:

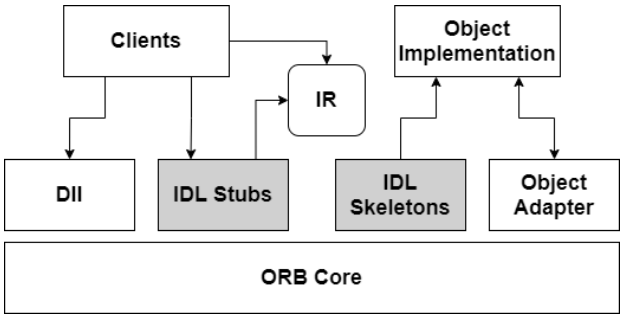

Fig. 3: CORBA main components

1) Object Request Broker (ORB) Core: CORBA consists of a set of objects that each provides services to the clients. ORB is the core system that is responsible for delivering requests from clients to objects and return any responses. ORB is important to provide transparency to the clients such that they do not have to know where the objects are, what the communication protocol is, or what the implementation of these objects is. Clients first have to hold a reference to the object then it can issue requests to that object using ORB core. When ORB core receives a request it will locate the object using the reference and activates it if it is not active then it will deliver the request to it. The object will execute the request and return the results to ORB and it will return it to the client [24].

2) Interface Definition Language (IDL): Object references are used to identify objects in the system but they do not specify what operations can be performed on that object. The client needs to know the interface for the object to know what requests it can send to it and what the expected responses are. OMG designed an interface definition language to describe object interfaces. IDL is declarative and language-independent. It separates the interface from the implementation which allows invoking operations from any programming language regardless of what the implementation of the object is or what language it is written in. The interface is used to generate compile-time stubs which are functions without an implementation that can be called from the client as a normal function. Stubs will take the parameters, convert them into a request, and send it to the target object [25].

3) Dynamic Invocation Interface (DII): Static stub using IDL is not the only way to invoke operations on objects. CORBA provides a way to dynamically invoke operations on objects that are not known at compile-time using the Dynamic Invocation Interface. A gateway, for example, does not need to be recompiled every time a new object is introduced. Instead, it can use DII to convert any request it receives into a dynamic dispatch and send it to the referenced object. DII can invoke operations synchronously using RPC-like style or deferred synchronous where the caller can specify whether to wait for the response or not [25].

4) Interface Repository (IR): The interface repository stores all IDL interfaces as runtime data structures and allows applications to access these interfaces and write them programmatically. IR is essential for DII to call methods on objects dynamically because it stores interfaces, methods, parameters, and response formats. Applications can use this information to traverse all interfaces and the methods inside them as well as all the types and describes all the operations supported by an object [25].

5) Object Adapter: Object Adapter is a layer between the CORBA implementation and the application. It provides the ORB interface according to the specification to allow any CORBA-based application to use it directly regardless of the CORBA implementation. Object Adapters include the ability to register object implementations, generate references for CORBA objects, activate server processes, activate objects, request demultiplexing, and object upcalls [25].

B. Fault-Tolerant CORBA

Many applications using CORBA require support for fault tolerance. These applications range from critical large-scale applications to medium or small non-critical applications that require high availability. Fault tolerance refers to eliminating all single-point of failure from the system. Fault Tolerance can be achieved by different strategies such as replication, request retry, load balancing, and immediate recovery [15].

C. Object Groups

One method to achieve fault tolerance is to create several replicas of the same object and group them as one unit (See Fig 4). Clients will invoke methods on the group and it will send the invocation to all member objects. Each object will run the invocation and return its result. Clients are unaware of the existence of multiple objects. As a result, if one object fails other objects that succeed will return the response without the client noticing the failure [16].

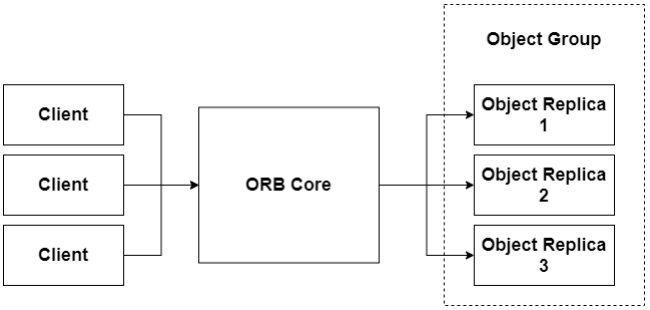

Fig. 4: Object replication using Object Groups

Landis and Maffeis [17] showed how to extend CORBA to support the features required for fault tolerance and reliability. They provided a detailed description of the requirements for

one-to-many communication between a client and an object group. They provided two examples that can be significantly simplified when implemented using the group object. The first example is a fault-tolerant directory service. The second is a reliable stock exchange ticker application. They showed how these examples might be implemented using object groups. They also analyzed two CORBA-compliant environments that implement fault-tolerant ORB: Electra and Orbix+Isis. They also analyzed two low-level requirements for implementing a fault-tolerant environment: Isis [18] and Horus [19].

Maffeis [20] designed and implemented a CORBA-compliant ORB environment (Electra) that permits the implementation of objects to be grouped into named unit. This group uses reliable multicast communication to share the operations between all the implementations in that group. Electra allows transparent communication where a group appears as one singleton object and allows non-transparent communication where an implementation can access the results of any invocations. Actions in Electra can be performed synchronously, asynchronously, or deferred-synchronously. Electra uses the underlying toolkit to provide constraints on the ordering of events where programmers can specify the requirements for when the invocations are dispatched. Electra is written in C++ by making two slightly modified interfaces: the BOA class and the Environment class.

D. Virtual Synchrony

Virtual Synchrony is a model in which it is guaranteed that the behavior of a distributed system is predictable even if a partial failure happens. When a multicast message is sent to a group of objects, virtual synchrony guarantees that either all objects receive the message, or no object receives it. It is never the case that some objects receive it and some do not because this will make the objects in the group in an inconsistent state [21].

E. Failure Detection

A reliable system requires the detection of when a failure happens because all objects in a group need to acknowledge requests. Without it, a client will block forever waiting for objects that failed to finish. The system will automatically check for failure by a timeout failure detector. The detector will consider any object that takes more time than a maximum limit to be failed and will notify other object in order to maintain consistency.

F. Message Queues

Maffeis and Olsen [22] proposed an easy way to achieve reliability using message queues. If process A wants to send a message to process B then process A will send the message to its queue handler, which is a separate process that will store the message in non-volatile storage. Then, the handler will attempt to send the message to process B's handler. If process B handler is online then it will delete the message from the queue, otherwise, it will repeatedly attempt to send it until process B's handler is online. This allows process A to send the message and forget about it and the handler is responsible for making sure the message is delivered.

G. Synchronization and Concurrency Control

In a distributed environment, there is often the need to access an object for a service from different objects at the same time and that will cause corruption to happen in that object. CORBA provides the specification for the Concurrency Control Service. The concurrency control service allows clients to acquire and release locks in two modes: transactional mode using the Transaction Service, and non-transactional mode on behalf of the current thread. When a client try to take a lock that is already acquired by another client, it will be forced to wait by blocking until the lock is released. This guarantees that only one client is using the resource at any time. There are three lock modes: Read, Write and Upgrade. The read and write lock supports the known policy which is multiple people are allowed read access concurrently but only one write access is allowed at any time. This means that either multiple people are reading at the same time or one write at a time but never read and write together or multiple writes. Upgrade Access is used to avoid a deadlock when multiple clients already have Read Access and want Write Access. Without Upgrade Access, they both will attempt to acquire Write Access and they both will deadlock forever. In this case, Upgrade Access can be used to denote that others can still have read access but not upgrade access or write access. This way if two clients want to read then write an object only one of them is allowed [26].

V. DIFFERENCES AND SIMILARITIES

The comparison of RPC, RMI and CORBA are listed in the Table I.

VI. CONCLUSION

There has been a huge focus on message passing for communication between components of the distributed system. Distributed systems are critical for many large-scale applications such as Air traffic control, defense systems, medical systems, telephony and networking systems, supply chains systems, stock exchange systems, etc. and message passing plays a significant role to ensure efficiency and reliability. This paper provides an overview of RPC technology and various implementations and the properties of each one. It provides an overview of RMI which an RPC implementation by Oracle, the advantages of using RMI, and various studies related to RMI. It also provides an overview of CORBA which a framework for distributed object computing designed by OMG, an overview of its different components, and an overview of the research on how to achieve reliability and fault-tolerance as an extension to CORBA.

REFERENCES

[1] T. Lindholm, F. Yellin, G. Bracha, and A. Buckley, "The Java® Virtual Machine Specification," p. 606.

TABLE I: Summarizes the similarities and differences between RPC, RMI, and CORBA.

	RPC	RMI	CORBA
What is it?	Remote Procedure Call is a protocol that allows one program to order a service from another program on a network without having to know the network's specifics.	RMI is an implementation of message passing mechanism. It allows Java virtual machines to communicate and invoke methods resides in other JVMs. It's a fundamental mechanism in distributed systems field.	CORBA is a standard not an implementation. It can be implemented by any vendor as a framework for distributed computing and it will guarantee interoperability with other implementations.
Operation System	Remote Procedure Call uses IDL it can use different operating systems and computer languages.	RMI can operates whereas it is a java platform.	CORBA is OS-independent, this means it can be implemented on any operating system allowing clients and servers on different operating systems to communicate with each others.
Programming Language	Remote Procedure Call Language (RPCL) is identical to the eXternal Data Representation (XDR) language.	RMI is a Java programing language package, located at java.rmi;	CORBA specification does not assume any programming language. It can be implemented using any programming language allowing them to communicate with each other.
Stubs	The stubs simulate a working local unit by hiding the code's "distance" on the other side. They also serve as process interfaces.	Stub in RMI is a class that implements the remote interface. It operates as a client-side representation for the remote object. The stub interacts with the server-side skeleton via the network.	CORBA use Interface Definition Language (IDL) to allow the application to specify the interface and CORBA will use it to generate the stubs in the programming language that the implementer is using.
Features	Batching is one of RPC's functions, allowing a client to send an arbitrarily large number of call messages to a server. A client may use broadcasting to transmit a data packet to the network and then wait for several responses. A server may become a client and render an RPC callback to the client's method using callback procedures.	RMI aimed to support Object-oriented programing in distributes system environment, it is a successor version of RPC and more efficient. One of the essential and special features of RMI is its capability to download the implementation of an object's class if it is not defined in the receiver's machine. All of the implementation and body of an object, before available only in one Java virtual machine, can be transferred to another, even remote, JVM.	CORBA is designed to be feature-complete specification. It includes the transport protocol, interface definition language, dynamic interface invocation, object adapters, interoperability, location transparency, synchronization and transactions.

[2] D. Hou and H. Xia, "Design of Distributed Architecture Based on Java Remote Method Invocation Technology," in 2009 International Conference on Environmental Science and Information Application Technology, Wuhan, China, Jul. 2009, pp. 618–621, doi: 10.1109/ESIAT.2009.235.

[3] T. Sysala and J. Janecek, "Optimizing remote method invocation in Java," in Proceedings. 13th International Workshop on Database and Expert Systems Applications, Aix-en-Provence, France, 2002, pp. 29–33, doi: 10.1109/DEXA.2002.1045872.

[4] Seong-Eun Chu, Dae-Wook Kang, and Jae-Nam Kim, "Efficient Implementations of Remote Method Invocation Considering Object Consistency," in 2006 International Conference on Hybrid Information Technology, Cheju Island, Nov. 2006, pp. 634–641, doi: 10.1109/ICHIT.2006.253558.

[5] G. L. Taboada, C. Teijeiro, and J. Tourino, "High Performance Java Remote Method Invocation for Parallel Computing on Clusters," in 2007 IEEE Symposium on Computers and Communications, Santiago, Portugal, Jul. 2007, pp. 233–239, doi: 10.1109/ISCC.2007.4381536.

[6] M. Lobosco, A. Silva, O. Loques, and C. L. de Amorim, "A New Distributed JVM for Cluster Computing," in Euro-Par 2003 Parallel Processing, vol. 2790, H. Kosch, L. Böszörményi, and H. Hellwagner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1207–1215.

[7] J. Yin, K. Agarwal, M. Krishnan, D. Chavarria-Miranda, I. Gorton, and T. Epperly, "Implementing High Performance Remote Method Invocation in CCA," in 2011 IEEE International Conference on Cluster Computing, Austin, TX, USA, Sep. 2011, pp. 547–551, doi: 10.1109/CLUSTER.2011.78.

[8] T. H. Dineen, P. J. Leach, N. W. Mishkin, J. N. Pato, and G. L. Wyant, "The network computing architecture and system: an environment for developing distributed applications," in Digest of Papers. COMPCON Spring 88 Thirty-Third IEEE Computer Society International Conference, San Francisco, CA, USA, 1988, pp. 296–299.

[9] A. S. Tanenbaum and R. van Renesse, "A Critique of the Remote Procedure Call Paradigm," p. 10.

[10] A. D. Birrell and B. J. Nelson, "Implementing Remote Procedure Calls," ACM Trans. Comput. Syst., vol. 2, no. 1, p. 21, 1984.

[11] Bacon and K.G. Hamilton, "Distributed Computing with the RPC: the Cambridge Approach", Distributed Processing, IFIP, North-Holland, 1988, pp. 355-369.

[12] K. Cheung, C. Chow, J. Koontz, M. Li, and B. Self "Project Athena Success in Engineering Projects" 6.933 Final Project Fall 1999.

[13] H. Bagci and A. Kara, "A Lightweight and High Performance Remote Procedure Call Framework for Cross Platform Communication:," in Proceedings of the 11th International Joint Conference on Software Technologies, Lisbon, Portugal, 2016, pp. 117–124.

[14] D. Zalewski, "Security Enhancement of Java Remote Method Invocation," in 2006 International Conference on Dependability of Computer Systems, Szklarska Poreba, 2006, pp. 223–231, doi: 10.1109/DEPCOS-RELCOMEX.2006.47.

[15] "Fault Tolerant CORBA", OMG.org, 2010. [Online]. Available: `https://www.omg.org/spec/FT/1.0/PDF`. [Accessed: 13-Mar- 2021]

[16] S. Maffeis, "The Object Group Design Pattern", COOTS, vol. 96, p. 12, 1996.

[17] S. Landis and S. Maffeis, "Building reliable distributed systems with CORBA", Theory and Practice of Object Systems, vol. 3, no. 1, pp. 31-43, 1997. Available: 10.1002/(sici)1096-9942(1997)3:1¡31::aid-tapo4¿3.0.co;2-a.

[18] F. Reynolds, "Reliable distributed computing with the Isis toolkit [Book Reviews]", IEEE Parallel and Distributed Technology: Systems and Applications, vol. 4, no. 3, p. 71, 1996. Available: 10.1109/mpdt.1996.532142.

[19] R. van Renesse, K. Birman and S. Maffeis, "Horus: A Flexible Group Communication System", Communications of the ACM, vol. 39, no. 4, pp. 76-83, 1996. Available: 10.1145/227210.227229..
[20] S. Maffeis, "Adding Group Communication and Fault-Tolerance to CORBA", Coots, vol. 95, pp. 135-146, 1995.
[21] K. Birman, "Reliable distributed systems". New York: Springer, 2010.
[22] S. Maffeis and D. Schmidt, "Constructing reliable distributed communication systems with CORBA", IEEE Communications Magazine, vol. 35, no. 2, pp. 56-60, 1997. Available: 10.1109/35.565656.
[23] S. Vinoski, "Distributed Object Computing With CORBA", C++ Report, pp. 32-38, 1993.
[24] S. Vinoski, "CORBA: integrating diverse applications within distributed heterogeneous environments", IEEE Communications Magazine, vol. 35, no. 2, pp. 46-55, 1997. Available: 10.1109/35.565655 [Accessed 14 March 2021].
[25] Z. Yang and K. Duddy, "CORBA", ACM SIGOPS Operating Systems Review, vol. 30, no. 2, pp. 4-31, 1996. Available: 10.1145/232302.232303 [Accessed 14 March 2021].
[26] "Concurrency Service Specification", Omg.org, 2000. [Online]. Available: https://www.omg.org/spec/CONC/1.0/PDF. [Accessed: 16- Mar- 2021].
[27] E. Tilevich and Y. Smaragdakis, "Portable and Efficient Distributed Threads for Java," in Middleware 2004, vol. 3231, H.-A. Jacobsen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 478–492.
[28] Barkly, J., "Comparing Remote Procedure Calls," NISTIR, 1993.
[29] P. G. Soares, "On remote procedure call," CASCON '92: Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research - vol. 2, p. 215–267 November 1992.
[30] B. N. Bershad, D. T. Ching, E. D. Lazowska, J. Sanislo and M. Schwartz, "A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems", IEEE Transactions on Software Engineering, vol. se-13, p 880-894, 1987.
[31] C. Pu, T. Autrey, A. Black, C. Consel, C. Cowan, J. Inouye, L. Kethana, J. Walpole and K. Zhang, "Optimistic Incremental Specialization: Streamlining a Commercial Operating System", ACM SIGOPS Operating Systems ReviewDecember, 1995, https://doi.org/10.1145/224057.224080
[32] G. Muller, R. Marlet, E. N. Volanschi, C. Cinsel and C. Pu "Fast, Optimized Sun RPC Using Automatic Program Specialization" IEEE Proceedings. 18th International Conference on Distributed Computing Systems (Cat. No.98CB36183), 1998, DOI: 10.1109/ICDCS.1998.679507.
[33] A. S. Tanenbaum and G. J. Sharp, "The Amoeba Distributed Operating System", Communications of the ACMDecember, 1990, https://doi.org/10.1145/96267.96281
[34] A. D . Birrell and B. J. Nelson, "Implementing remote procedure calls", ACM Transactions on Computer Systems, February 1984.
[35] J. W. Stamos and D. K. Gifford, "Remote evaluation", ACM Transaction s on Programming Languages and Systems, vol. 4, p. 537-565, October 1990.
[36] M. Satyanarayanan,"zIntegrating security in a large distributed system", ACM Transactions on Computer Systems, vol. 3 p. 247-280, August 1989.