6/8/2020

Sumanth Mohan, Narinder Pal Singh

Lambton college in toronto

Assignment 2

Embedded Middleware Distributed systems using CORBA

**GROUP -K**

**Sumanth Mohan - C0752365**

**TABLE OF CONTENTS:**

1. **Introduction**
2. **Design of Embedded Middleware distributed systems**
3. **Architecture of Embedded Middleware distributed systems**
4. **Networking of Embedded Distributed systems**
5. **Internetworking of Embedded Distributed systems**
6. **InterProcess Communication**
7. **Remote Method Invocation in Middleware systems**
8. **Remote Procedure Call in Middleware Systems**
9. **Difference Between RMI and RPC in Middleware systems**
10. **Indirect Communication**
11. **Security Middleware**
12. **Mobile and Ubiquitous Computing**

**Abstract**

**In this research paper, we discuss the embedded middleware systems in detail. The usage of Embedded middleware in improving the efficiency, performance and flexibility of the Distributed middleware architecture is discussed in detail. The Embedded Middleware is a standardised object-oriented application interface for supporting distribution of networked embedded applications, would solve the networking problems of communication applications. A working example of one of the latest technologies in the Middleware architecture CORBA has also been introduced and explained in detail in this research paper. Various topics such as Design, Networking and Internetworking, InterProcess Communication, Remote Invocation, Indirect Communication, Security Middleware, Distributed File Systems, Mobile and Ubiquitous Computing are discussed in this research paper. It is to be noted that all the content used in this research paper have been professionally cited and referenced.**

***Keywords: middleware software, embedded software, embedded system. RPC, RMI, Distributed System***

1. **Introduction**

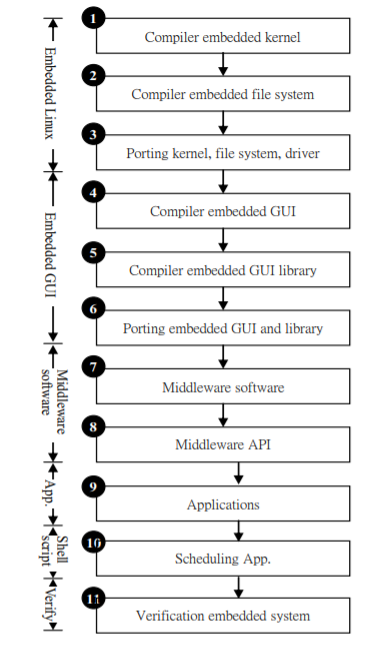
With Enterprise Middleware getting bigger and better each day, it is important to have a standardised middleware solution to every existing problem. This would ensure the systems are highly scalable, compatible as well as customer oriented. Embedded systems with communicating and computing ability and multimedia functions work to every corner of daily life ***(Reference 1, 2012)*[1].** However, the diverse architectures of embedded systems cause problems corresponding to reuse, portability and dependability.

Middleware is a set of software that executes between operating system and application to solve stated problems. The advantages include unified interface, scalable and transparent abilities. This project investigates middleware technology on embedded systems and then proposes embedded middleware architecture to overcome the problems relating to reuse, portability, dependability and transparency. Heterogeneous system architectures are currently the main platform on which an ever-increasing number of innovative applications (i.e. smart home or ambient intelligence applications) rely. When designing these complex systems, one of the most time-consuming tasks is the definition of the communication interfaces between the different components through a number of scattered heterogeneous processing nodes ***(Reference Paper2, 2006)***[**2]**. That is not only a complex task, but also very specific for a particular implementation, which may limit the flexibility of the system, and makes the solutions difficult to reused. Some concepts taken from distributed object platforms such as CORBA or Java RMI have already been applied to SoC design in order to get a unified view of HW and SW modules. CORBA does provide an Interface Description Language to facilitate both service development and service integration. IDL is a

neutral, portable specification language, which compilers can map to other languages (C++ and Java) to provide executable code. We can use this code to integrate different management protocols and systems. When designing these complex systems, one of the most time-consuming tasks is the definition of the communication interfaces between the different components through a number of scattered heterogeneous processing nodes

1. **Design**

In this section we discuss the Design of the Embedded distributed systems. The Embedded system design is designed based on several components integrated and put together to form a formation. These components enhance the working and the configuration of the middleware architecture.

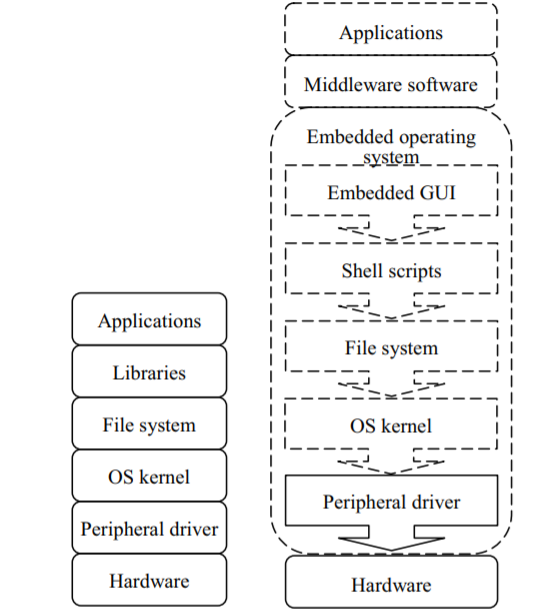


***Figure 1: Design of the Embedded Middleware Distributed system***

The above Figure shows design of the Embedded middleware architecture systems. This figure shows the design flow of middleware software. First, we compile the embedded kernel for attaining embedded operating system ***(Reference 1, 2012)*[1].** In step 2, we produce the embedded file system by compiling the root file system. In step 3, embedded kernel, file system and driver are verified via porting procedure to embedded platform. In step 4, the embedded graphical user interface (GUI) gives an interactive environment to user. It is more familiar to user than embedded text environment. On the other hand, the libraries of embedded GUI are provided in step 5 and 6 while designers develop embedded GUI applications. In step 7, we develop middleware software that comprises of service manager and content manager sub-modules. The middleware application programmable interfaces (APIs) are done in step 8. The proposed of middleware software are verified by two embedded systems. Those applications are designed in step 9. Scheduling applications procedures are deal on step 10. Finally, the whole embedded systems with middleware software are verified in step 11.

**3. Architecture**

In this section we discuss a very important and critical part of this report which is the architecture of the Embedded distributed system.



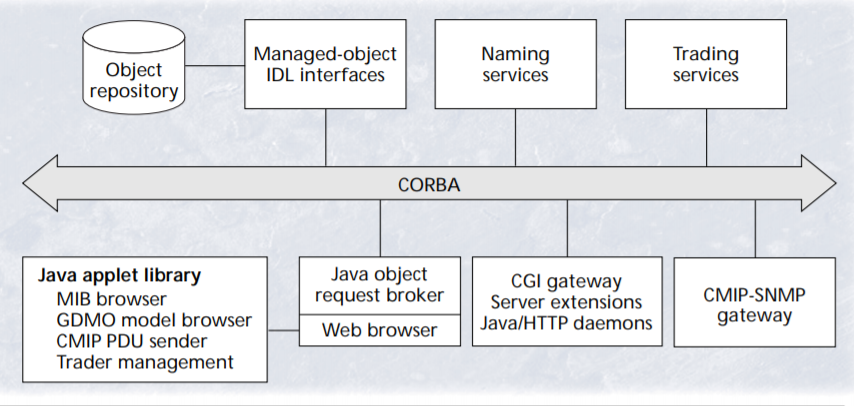
***Figure 2: Architecture of Embedded Middleware Architecture***

The middleware software divides into three sub-modules that comprise middleware application programmable interface, service manager as well as content manager. The middleware application programmable interface sub-module provides unified interface for upper module (i.e. application module). On the other hand, it supplies some methods such encode, decode or file function, etc. for lower sub-module inside middleware software ***(Reference 1, 2012)*** **[1].** Furthermore, it designs for solving for portability and dependability issues via a set of application programmable interfaces. Another sub-module namely service manager can automatic generation and deployment content for requirement of application. This sub-module also aims flexibility topic to provide related content for various applications. Moreover, it can automatic parse the structure of contents.

Also, the verification and completion for content are verified in service manager sub-module. Furthermore, it achieves scalability via auto deployment content. In short, this sub-module designs for conquering flexibility and scalability for various embedded systems. The other submodule called content manager develops for conducting content information. We design the format of content information for the purpose of solving transparent issue. The characteristics of content information include structural, self description and readable text and simple syntax.

**4. Networking of Embedded Distributed systems**

**Definition:** Networks for embedded computing span a broad range of requirements; many of those requirements are very different from those for general-purpose networks. Some networks are used in safety-critical applications, such as automotive control. Some networks, such as those used in consumer electronics systems, must be very inexpensive. Other networks, such as industrial control networks, must be extremely rugged and reliable. In this research Paper we explain the process of networking using CORBA ***(Research Paper 3, 1997)*[3].**

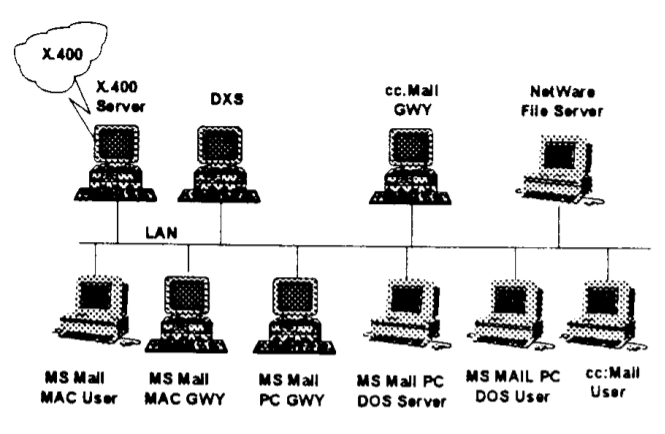


***Figure 3: Networking of Embedded distributed systems using CORBA***

A Java-based object request broker is an attractive concept because it combines advantages provided by CORBA and Java. CORBA provides scalable distributed services and is especially suitable for legacy application integration. The portability of Java source code and bytecode make it suitable for distributed computing. Iona’s OrbixWeb object request broker provides robust support for integrating Java and CORBA. A browser such as Netscape Navigator uses a very restricted security policy. An applet can connect via a socket only to the machine from which it was loaded. This thus restricts the CORBA daemon and servers to locations within the same machine from which the applet was loaded. Otherwise, a proxy must be used. version of X.500 names to CORBA names. The trading service provides another mechanism for dynamically finding distributed services. Service providers can advertise service offers to a trader. Potential service users (importers) can search information on available services and their accessibility. An importer may use the standard constraint language to restrict the properties of service offers. After a successful match, the importer can interact with the service provider. The next phase of DCP prototype will use the trading service in the intelligent network service management.

**5. Internetworking of Embedded Distributed systems**

In this section we discuss the topic of Internetworking in the Embedded Distributed systems. Internetworking is an integral part of the Embedded distributed system. It is important to design and Implement the embedded the networking keeping in mind various aspects which we will discuss in this section ***(Research paper 4, 1994)*[4].**

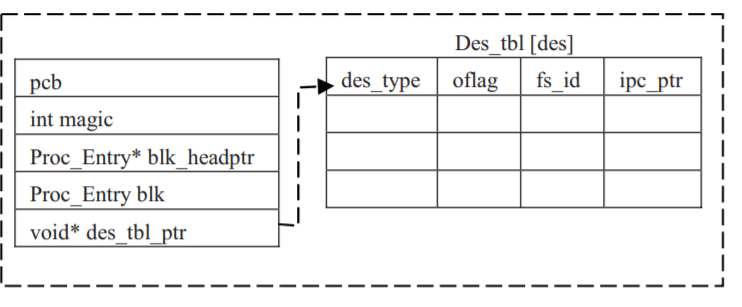


***Figure 4: Inter-networking in Embedded Middleware systems***

The goal architecture for an enterprise network is shown in Figure 4 . This architecture contains the functions described in both the RMODP and OSF's DCE. In order to fully integrate LANs and WANs into an enterprise network one requires interoperable services for communications, graphics, APIs, messaging, naming, etc. Impacting all these services are the overarching security and management services. Cost effective security and management can only be achieved when these other functions have been standardized. The goal of achieving interoperability through the use of mandated IS0 standards has not been achieved in the real world.

**6. InterProcess Communication**

In this section we discuss the very important topic of Interprocess communication in the Embedded distributed systems. The interprocess communication is very important in establishing a direct/indirect communication between the middleware systems ***(Reseach paper5, 2009)*[5**].

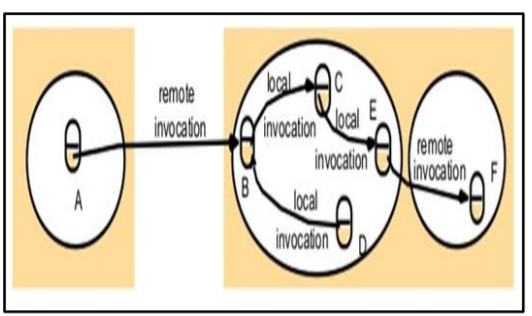


***Figure 5: Interprocess communication in Embedded distributed systems***

An efficient message queue mechanism is used for exchanging messages between different processes and the semaphore, which provides a protection scheme for the shared resources. The message queue and semaphore have different features and purposes. However, they share common structures, including the magic number, free list, lock, and waiting queue. Figure 1 displays the structure of a user process containing the necessary fields of the realtime interprocess communication as well as the normal process control block PCB. In this design, IPC fields are not embedded into the PCB and IPC is independent of the kernel design. Therefore, IPC and kernel can be designed in parallel without interfering with each other. pcb is a process control block structure designed by the kernel developer. magic is initially checked by every operation on RT\_IPC\_PCB to ensure it is working on the correct structure. The semaphore and the message queue have a waiting queue and each node of the queue is a Proc\_Entry blk\_headptr structure containing a previous and next pointer to the same structure as well as a PCB pointer. blk holds a Proc\_Entry structure and is used to link its own process with other processes when blocked. Some IPC operations deal with descriptors. A descriptor table, des\_tbl[], pointed by des\_tbl\_ptr pointer, is established to maintain related information associated with it.

**7. Remote Method Invocation in Middleware systems**

**Definition:** RMI (Remote Method Invocation) 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. In this section we discuss the Remote Method Invocation(Research paper 6, 2019) **[6]**.

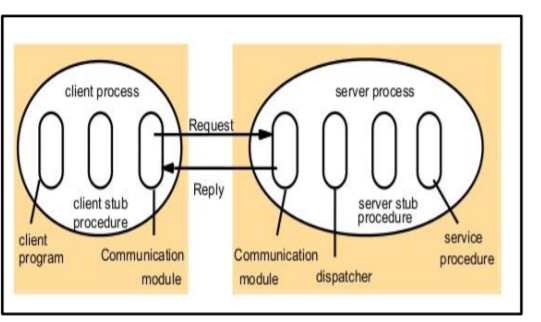


***Figure 6 : Remote Method Invocation in Middleware systems***

The remote method invocation is similar to RPC additional RMI = RPC + object orientation. Every procedure includes compilations of object which could receive a limited chant only. The RMI is a technique of invocation among objects with dissimilar processing in the same computer or different computerThe interface description dialect (IDL) will provide by CORBA framework which utilize the characterization remote interface. The remote articles kinds and customer projects may be actualizing in several dialects. For example, C++, Java or Python in IDL complier is accessible. The COBRA customers require not utilize the same dialect as the remote protests keeping in mind the end goal to conjure its systems remotely . Nonetheless in the distributed issue, the objects intricate in a series of associated calls may be addressed over various procedures. Once, call reaches the process, RMI Istaking role, in addition, the reference of remote Object required existing demander. The object A required holding the remote reference object of object B as illustrated in Figure 4. The remote reference object could be gained as accused of remote technique calls. Additionally, in Figure 4, the object A acquires a reference of a remote in F as of B object Several applications used the Garbage Collection (GC) notation that is used to terminate unused application inside memory at running time, to take advantage of releasing memory index, however, GC depending on index counts.

**8.Remote Procedure Call In Middleware Systems**

**Definition** : A remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

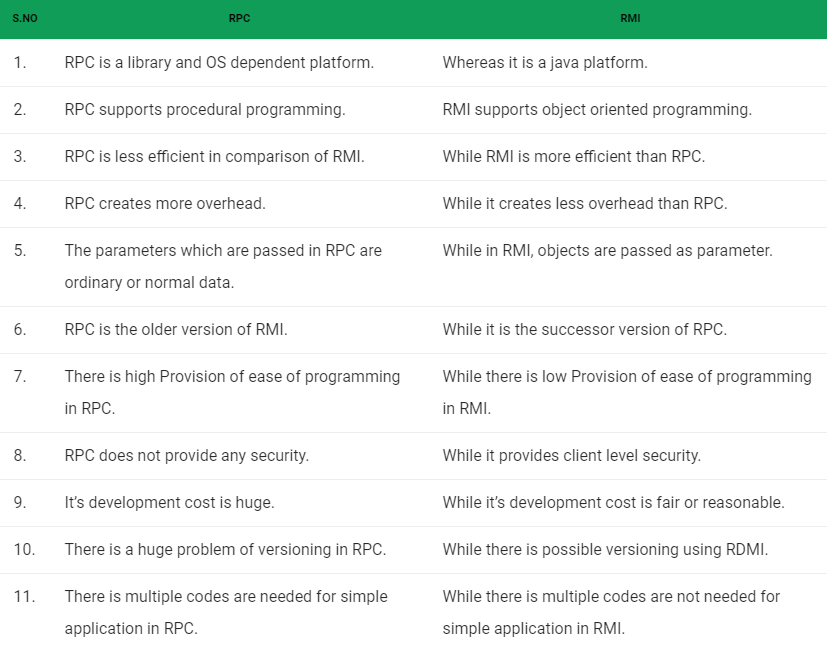


***Figure 7 : Remote Procedure Call In Middleware Systems***

The similarity of RMI with remote procedure call in term of client programming could permit to the server process for running in other programs. To provide sequence of RPC, any server could be client by additional server. Hence, the server progression will define in the interface of services and the viability of methods for remote invoke. Actually, the service is relatively alike to single one which had attributed. Hence, it does not provide the remote techniques ***(Research paper 6, 2019)*[6].** Thought, the RPC and RMI could be implemented to achieve new options of call. In general, the RPI could apply in the protocol of replay request with short elimination of location from the message requested as similar replay in case of neglected the reference field. Subsequently, procedure invoke is careless by mean of references of object and the objects itself. The incoming clients to the server contain one process of stub to every method of service interface. Hence, the regulation of stub method is close to proxy form which performs as limited technique with client. So, the place running invoke is marshal methods is individuality and complain with the message send via server connection equipment as none marshals repetitions. For more capability of interface, the server steps consist a sender with stub and one service process for every method. Corresponding to process ID in the demand message, the sender will choose one server stub methods. The server stub process looks like close to skeleton technique in none marshals form which complains with demand message. The achievements of service process could be summarizing as in and this includes RPC call abstraction procedure among much process, realization server procedure of sub client side of proxy and the marshal parameters locate sub client side.

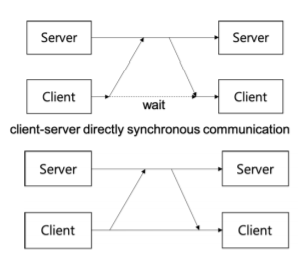
**9.Difference Between RMI and RPC in Middleware systems**

Below are the differences between the RMI and RPC in the Middleware distributed systems architecture ***(Reference paper 8, 2019)*[8]**



**10.Indirect Communication**

In this section we discuss the process of Indirect communication in the Embedded middleware distributed systems. There can be two types of Indirect communication namely, the Synchronous vs. asynchronous communication ***(Research paper 7, 2019)*** **[7].**

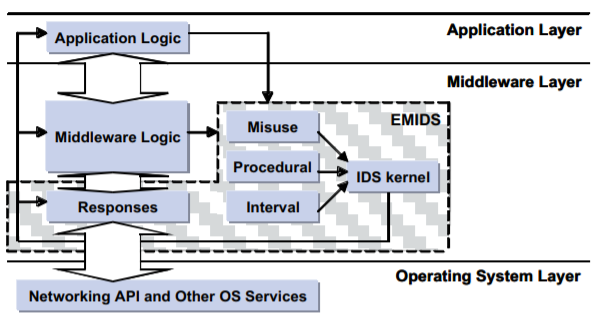


***Figure 8. Indirect Communication in Embedded distributed systems***

Most distributed system applications, such as online games, social networking services, cloud services, and IoT, communicate with each other using a client-server or peer-topeer model of communication structure. Communication functions required by the application can be developed by the application developer, but the application development efficiency can be improved by using the service of the communication middleware instead. Existing communication middleware and framework provide various communication services specialized for general purpose or a specific target application. In a client-server application, the communication method between the client and the server usually follows the request-reply pattern where a pair of the client’s request message and the server’s response message composes a communication service. The communication between the server and the client can be classified as synchronous and asynchronous.

**11.Security Middleware**

In this section we discuss another Important topic of Enterprise Middleware architecture which handling the security of the Middleware architecture(Research paper 9, 2005) ***[9]****.*



***Figure 9: Security Architecture of the Enterprise Middleware***

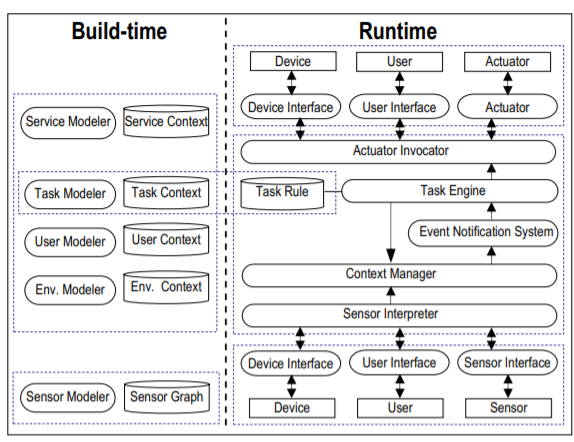
Data collection within the EMIDS model is performed by sensors embedded into the application or middleware framework, as illustrated in Figure 9. The data is processed by an IDS-kernel, which: 1. Provide a high level of configurability, which can be extended to support dynamic reconfiguration of the security policies or responses. Support multiple types of sensors and responses through a generalized interface. EMIDS sensors are integrated into the application or middleware layer, as illustrated in Figure 9. The different sensor types are discussed in depth in the next section. Based on the outcome of the data analysis or IDS-kernel configuration, the appropriate response will be triggered. Responses can be used to target the middleware connections or the application itself.

**12.Mobile and Ubiquitous Computing**

**Definition:** Mobile and Ubiquitous Computing publishes work focused on all aspects of mobile, wearable, pervasive, and ubiquitous computing.

Embedded systems have become ubiquitous ***(Research paper10, 2005)*[10].** New devices and systems have emerged, such as cellular phones, PDAs, and wireless networks. Older technologies also reap the benefits of embedded processing, for example a typical automobile now. The different sensor types are discussed in depth in the next section. Based on the outcome of the data analysis or IDS-kernel configuration, the appropriate response will be triggered. Responses can be used to target the middleware connections or the application itself.

In the ubiquitous environment, various devices can be used. When a service is provided, the service needs to be bound to given environment based on the information related to user’s intension. Insertion or deletion of a binding by service creation or environmental change must be considered



***Figure 10:*** ***Mobile and Ubiquitous Computing***

1) Service Modeler: As to model services to be invoked from a task, it defines services to be needed in application domain. Also, it consists of components for the implementation and management of defined services. 2) Sensor Modeler: As to collect the contextual information from the environment that services are performed and to deliver the information to a context manager, it is to model services to be provided actively. 3) Environment Modeler: The cyber space is an abstract model of the physical space. Environment is an abstract model of the place in

a specific domain. Users perform their work by interacting with the space and the environment where they are located through CASM. Environment modeler models not only sensors, devices, and services in environments but also hierarchically the environment for a domain

**Conclusion**

In this research paper, we have discussed the importance of Middleware architecture and how the Embedded middleware systems is playing a pivotal role in the Middleware architecture. With large processing of data and with the advent of operating systems across multiple platforms it is extremely important to have an architecture which is functional and fully efficient. The Embedded Middleware is a standardised object-oriented application interface for supporting distribution of networked embedded applications, would solve the networking problems of communication applications.

A working example of one of the latest technologies in the Middleware architecture CORBA has also been introduced and explained in detail in this research paper. Various topics such as Design, Networking and Internetworking, InterProcess Communication, Remote Invocation, Indirect Communication, Security Middleware, Distributed File Systems, Mobile and Ubiquitous Computing are discussed in this research paper. It is to be noted that all the content used in this research paper have been professionally cited and referenced

The Entire research paper has followed the standard IEEE level of formatting and all papers have been referenced, cited per the IEEE notation standards.

**References**

**[1]** Middleware Software for Embedded Systems, Yang-Hsin Fan, National Taitung University - <https://ieeexplore.ieee.org/document/6185100>

**[2]** System-Level Middleware for Embedded Hardware and Software Communication, Fernando Rincon, School of Computer Engineering - <https://ieeexplore.ieee.org/document/4408501>

**[3]** Java- and CORBA-Based Network Management, Mika Leppinen, Nokia Research Center - <https://ieeexplore.ieee.org/document/587555>

**[4]** LANNVAN Integration: Internetworking + Application Interoperability , E J Wells - <https://ieeexplore.ieee.org/document/473939>

**[5]** Embedded Systems with Improved Interprocess Communication Design, Hosseini Marzi, Department of Information Systems - <https://ieeexplore.ieee.org/document/5195803>

**[6]** A Comparative Study in Remote Techniques and Event-Based Invocations, Hamood Shehab Hamid, Faculty of Information Science and Engineering, Management & Science - <https://ieeexplore.ieee.org/document/8837063>

**[7]** Directly and Indirectly Synchronous Communication Mechanisms for Client-Server Systems Using Event-Based Asynchronous Communication Framework, MINGYU LIM, Department of Smart ICT Convergence - <https://ieeexplore.ieee.org/document/8744225>

**[8]** Difference between RPC and RMI, MKS075 - <https://www.geeksforgeeks.org/difference-between-rpc-and-rmi/>

**[9]** Configurable Middleware-Level Intrusion Detection for Embedded Systems, Deborah A. Frincke, Washington State University - <https://ieeexplore.ieee.org/document/1437169>

**[10]** A Middleware for Supporting Context-Aware Services in Mobile and Ubiquitous Environment, , Kang-Woo Lee, Electronics and Telecommunications Research Institute - <https://ieeexplore.ieee.org/document/1493688>