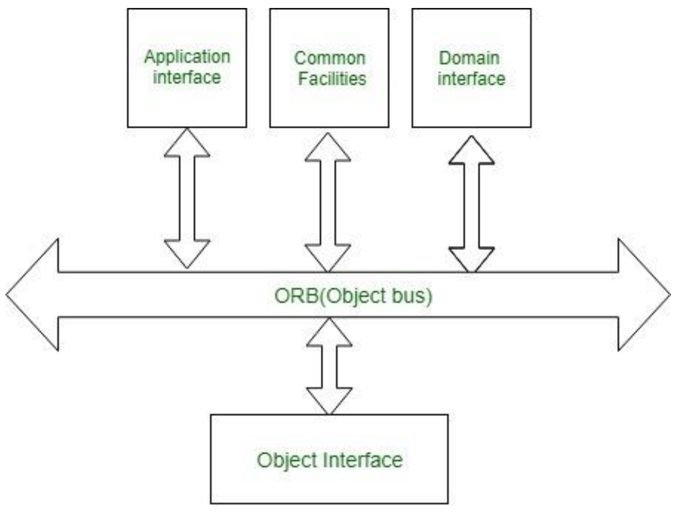
Distributed Computing

Experiment – 9

Case Study: CORBA	
Learning Objective:	Describe the concepts of distributed File Systems with some case studies
Learning Outcome:	Ability to perform case study on distributed file system.
Course Outcome:	**CSL801.6**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO11)Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.** **(PO12) Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Bloom's Taxonomy Level:	understanding
Theory:	Case Study: Common Object Request Broker Architecture (CORBA) **1. Introduction** Common Object Request Broker Architecture (CORBA) stands as a resilient and extensively utilized middleware technology, streamlining communication among software components dispersed across various platforms and programming languages. This segment furnishes an insight into CORBA's genesis, chronicling its evolutionary journey and underscoring its pivotal role in the realm of distributed computing.



Object Management Architecture(OMA)

Overview of Common Object Request Broker Architecture (CORBA)

CORBA emerges as a middleware standard conceived by the Object Management Group (OMG) to foster seamless interaction among distributed objects operating in heterogeneous computing environments. Central to CORBA's functionality is its utilization of an Object Request Broker (ORB) to orchestrate communication between these distributed entities, effectively shielding developers from the intricacies of network protocols and language discrepancies.

2. Background

Need for Middleware in Distributed Systems

The necessity for middleware in distributed systems lies in its role as an intermediary layer between disparate software components or systems, offering a standardized mechanism for communication, integration, and management. In distributed environments, where components may vary in programming languages, hardware platforms, or network settings, middleware is pivotal in facilitating interoperability and abstracting the intricacies of low-level communication.

Middleware delivers a range of services, including message queuing, remote procedure calls, object-oriented communication, transaction management, and security enforcement. By concealing the complexities of distributed communication and presenting a consistent interface for application components, middleware streamlines the development, deployment, and maintenance of distributed systems.

The progression of middleware technologies dates back to the early stages of distributed computing, where remote procedure call (RPC) mechanisms like Sun RPC and Distributed Computing Environment (DCE) RPC emerged to facilitate communication between distributed components. However, these initial middleware solutions often grappled with limitations in interoperability, platform independence, and scalability.

Recognizing the necessity for a standardized middleware solution capable of addressing the challenges of distributed computing more effectively, the Object Management Group (OMG) embarked on the development of the Common Object Request Broker Architecture (CORBA) in the early 1990s. CORBA's inception in 1991 with the release of CORBA 1.0 marked a significant milestone in the evolution of distributed computing. By offering a platform-independent, language-neutral middleware specification, CORBA transformed the landscape, establishing a standardized interface for distributed objects and introducing the concept of an Object Request Broker (ORB) to oversee communication between objects, thereby laying the groundwork for a new era of distributed computing.

3. Key Concepts of CORBA

3.1. Object Request Broker (ORB):

The Object Request Broker (ORB) serves as the central component in CORBA architecture. It facilitates communication between distributed objects by receiving requests from clients, locating the appropriate objects or services, and dispatching requests to the corresponding objects. ORB abstracts the complexities of network communication, language bindings, and platform differences, allowing distributed objects to interact seamlessly regardless of their implementation details.

3.2. Interface Definition Language (IDL):

The Interface Definition Language (IDL) is a specification language used in CORBA to define the interfaces of distributed objects. IDL provides a platform-independent and language-neutral way to describe the operations and data structures supported by objects in a distributed system. IDL specifications are compiled into language-specific stubs and skeletons, which enable clients and servers written in different programming languages to communicate with each other transparently.

3.3. Object Adapter:

The Object Adapter acts as an intermediary between the ORB and the distributed objects within a CORBA application. It manages the lifecycle of objects, including instantiation, activation, deactivation, and destruction. The Object Adapter also provides mechanisms for object activation policies, which determine how objects are created and managed in response to client requests.

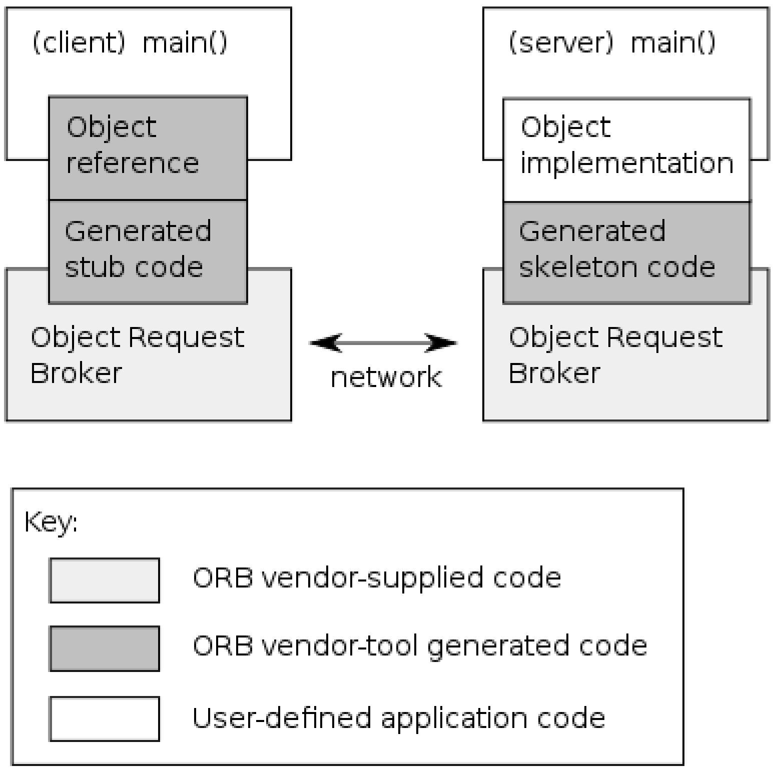
3.4. Object Services:

Object Services are a set of standardized services provided by CORBA to support common distributed computing tasks. These services include naming and directory services, security services, transaction services, event services, concurrency control services, and persistence services. By leveraging these pre-defined services, developers can build distributed applications more efficiently without having to implement these functionalities from scratch.

3.5. Portable Object Adapter (POA):

The Portable Object Adapter (POA) is an enhanced version of the Object Adapter introduced in later versions of CORBA. POA provides additional features and flexibility for managing distributed objects, including support for object activation policies, multiple object implementations, and location transparency. POA enhances the scalability and performance of CORBA applications by allowing developers to fine-tune the behavior of distributed objects based on their deployment requirements.

4. Implementation and Architecture



CORBA (Common Object Request Broker Architecture) employs a distributed architecture comprising several key components that work together to facilitate communication between distributed objects. At the heart of CORBA is the Object Request Broker (ORB), which acts as a middleware layer responsible for managing communication between clients and servers.

Components Involved in CORBA

1. **Object Request Broker (ORB):** At the heart of CORBA lies the ORB, serving as its central component. It assumes responsibility for the discovery of objects, execution of methods on remote objects, and supervision of communication between clients and servers.
2. **Interface Definition Language (IDL):** IDL serves as the lingua franca for defining the interfaces of CORBA objects in a manner that transcends specific programming languages and platforms. These IDL specifications undergo compilation into language-specific stubs and skeletons, facilitating seamless communication between clients and servers.
3. **Object Adapter:** The Object Adapter assumes the mantle of managing the lifecycle of CORBA objects. It oversees tasks such as object instantiation, activation, deactivation, and garbage collection. By acting

as an intermediary between the ORB and application objects, it ensures that their interfaces align with the CORBA standard.

4. **Object Services:** Within the CORBA framework, a suite of standard Object Services is provided. These services furnish distributed objects with common functionalities including naming, security, transactions, concurrency control, and event notification. By offering reusable components for these routine tasks, Object Services streamline the development of distributed applications.
5. **Portable Object Adapter (POA):** Representing an advanced iteration of the Object Adapter, the Portable Object Adapter (POA) extends augmented support for object activation, delegation, and location transparency. Through dynamic activation and deactivation based on demand, POA enhances scalability and optimizes resource utilization for CORBA objects.

Communication Protocols Used in CORBA

CORBA introduces diverse communication protocols to facilitate the exchange of requests and responses between clients and servers. Among the prevalent communication protocols utilized in CORBA, the following stand out:

1. **Internet Inter-ORB Protocol (IIOP):** IIOP emerges as a standardized protocol pivotal for facilitating communication among Object Request Brokers (ORBs) within distributed CORBA systems. Rooted in the Internet's TCP/IP protocol suite, IIOP fosters interoperability among CORBA implementations across varying vendors' offerings.
2. **GIOP (General Inter-ORB Protocol):** GIOP serves as a protocol instrumental in encapsulating CORBA requests and responses, facilitating their transmission across networks. It delineates the structure and format of messages exchanged between ORBs, thereby furnishing a uniform communication protocol indispensable for CORBA-based applications.

Interaction Between Clients and Servers in CORBA Architecture

In the typical architecture of CORBA, the interaction between clients and servers follows a structured sequence of steps:

1. **Client Invocation:** Initiating the interaction, the client triggers a method call on a remote CORBA object by utilizing a local proxy object.
2. **Stub Invocation:** Subsequently, the client's method call is intercepted by a stub, which encapsulates and marshals the method parameters into a request message. This message is then dispatched to the Object Request Broker (ORB).
3. **ORB Processing:** Upon receiving the request message, the ORB undertakes the task of locating the target object based on its object reference. Once identified, the ORB forwards the request to the appropriate server.
4. **Skeleton Invocation:** On the server side, the ORB receives the request and dispatches it to the corresponding skeleton object. The skeleton then unmarshals the method parameters and invokes the actual method on the server object.
5. **Server Processing:** Executing the requested method, the server processes the operation and generates the result, which is subsequently returned to the client via the reverse path.
6. **Response Propagation:** The server's ORB marshals the method result into a response message, transmitting it back to the client's ORB through the same communication channel employed earlier.
7. **Stub Processing:** Finally, the client's ORB receives the response message, unmarshals the result, and delivers it to the client application through the local proxy object, thus completing the interaction cycle.

5. Advantages of CORBA

1. **Interoperability:** CORBA excels in interoperability, fostering seamless communication between objects developed in diverse programming languages and operating on different platforms. This is achieved through standardized interfaces defined using the Interface Definition Language (IDL), ensuring CORBA objects interact harmoniously regardless of implementation specifics.

2. **Language and Platform Independence:** Embracing language and platform independence, CORBA empowers developers to construct distributed systems using their preferred languages and environments. By establishing a standardized communication infrastructure via the Object Request Broker (ORB), CORBA liberates systems from language or platform constraints, enabling transparent communication across heterogeneous environments.
3. **Reusability and Scalability:** CORBA champions component reusability and scalability, enabling organizations to construct intricate systems by amalgamating pre-existing CORBA components. Through IDL-defined interfaces and standard object services, CORBA fosters the creation of modular and reusable components, seamlessly integrable into larger systems. Additionally, its support for distributed object activation and delegation facilitates dynamic scalability to accommodate burgeoning workloads.
4. **Support for Distributed Computing:** Tailored for distributed computing environments, CORBA provides robust support for constructing distributed systems spanning networked landscapes. Leveraging features like distributed object activation, remote method invocation, distributed transactions, and object services such as naming and event notification, CORBA streamlines the development and operation of distributed applications within networked environments.
5. **Integration Capabilities:** CORBA boasts robust integration capabilities, enabling seamless integration of disparate systems and technologies into a cohesive distributed architecture. By furnishing standardized interfaces and communication protocols, CORBA facilitates integration with legacy systems, third-party components, and heterogeneous environments. This integration prowess fosters interoperability between new and existing systems, empowering organizations to capitalize on their current investments while embracing novel technologies.

6. Challenges and Limitations

1. **Complexity of CORBA Implementations:** The intricate nature of implementing CORBA presents a significant challenge. Developing CORBA-based applications demands a profound grasp of distributed computing principles, intricate ORB configurations, IDL

specifications, and other specialized CORBA technologies. Managing this complexity can be daunting, particularly for developers with limited experience in distributed computing.

2. **Performance Overhead:** CORBA introduces performance overhead owing to the additional layers of abstraction and communication protocols involved in remote method invocations. Tasks like marshalling and unmarshalling data, network communication, and ORB processing contribute to latency and diminished throughput in CORBA-based systems. Although optimizations like object activation and connection pooling can alleviate some performance concerns, CORBA may exhibit higher latency compared to leaner communication mechanisms.
3. **Security Concerns:** Security emerges as a critical concern in distributed systems, and CORBA implementations are susceptible to security vulnerabilities. Insecure communication channels, inadequate authentication mechanisms, and lax access control policies can compromise the confidentiality, integrity, and availability of data exchanged over CORBA networks. Safeguarding against these threats necessitates meticulous configuration of security features such as SSL/TLS encryption, robust authentication mechanisms, and stringent authorization policies.
4. **Vendor Lock-In:** The prospect of vendor lock-in poses a potential limitation for CORBA users. Organizations may become reliant on a specific vendor's CORBA implementation, making transitions between different implementations or migrating away from CORBA challenging and costly. This dependency, compounded by reliance on vendor-specific extensions or features, can impede flexibility and hinder the exploration of alternative middleware solutions or emerging technologies.
5. **Compatibility Issues:** Compatibility discrepancies may arise within CORBA environments due to variations in implementations, versions, and vendor-specific extensions. Incompatibilities between ORBs, IDL compilers, and object adapters can precipitate interoperability issues, complicating the integration of components developed across different CORBA implementations or system upgrades. Mitigating compatibility concerns necessitates meticulous planning, meticulous version control, and adherence to standardized CORBA specifications.

7. Use Cases and Applications

Examples of Industries Leveraging CORBA:

- **Telecommunications:** In the telecommunications sector, CORBA finds widespread application in network management, billing systems, and service provisioning, enabling efficient and reliable operations.
- **Finance:** Financial institutions rely on CORBA for critical functions such as transaction processing, risk management, and trading systems, ensuring secure and streamlined financial operations.
- **Healthcare:** Within healthcare, CORBA plays a vital role in electronic health record (EHR) systems, medical imaging, and patient management applications, enhancing healthcare delivery and patient care.
- **Aerospace and Defense:** CORBA is integral to aerospace and defense industries for command and control systems, simulation, and mission-critical applications, ensuring robust and reliable operations in challenging environments.

Real-World Applications of CORBA:

- **Enterprise Integration:** CORBA facilitates seamless integration between disparate systems, enabling organizations to communicate and exchange data efficiently, thereby streamlining operations and enhancing productivity.
- **Distributed Computing:** In distributed computing environments, CORBA enables the development of scalable and interoperable systems across networked environments, facilitating efficient resource utilization and improved system performance.
- **Web Services:** CORBA serves as a robust middleware platform for implementing web services and service-oriented architectures (SOA), enabling organizations to build flexible and scalable distributed systems that can adapt to changing business requirements.
- **Grid Computing:** In grid computing environments, CORBA is utilized for resource management, job scheduling, and data sharing across distributed nodes, enabling organizations to harness computational resources efficiently for large-scale scientific and computational tasks.

Case Studies Showcasing Successful Implementations of CORBA:

- **Telecommunications Case Study:** A leading telecommunications company implemented CORBA for real-time network management. By leveraging CORBA, the company achieved improved scalability and performance in managing its network infrastructure, enabling it to handle increasing traffic demands efficiently while ensuring uninterrupted service delivery to customers.
- **Financial Services Case Study:** A prominent banking institution deployed CORBA for transaction processing, facilitating seamless integration between its core banking systems and third-party applications. CORBA's robust middleware architecture enabled the bank to streamline transaction processing, enhance data accuracy, and improve overall operational efficiency, thereby enhancing customer satisfaction and driving business growth.
- **Healthcare Case Study:** A reputable hospital adopted CORBA for its electronic health record (EHR) system, aiming to enhance data interoperability and patient care delivery. By implementing CORBA, the hospital achieved seamless integration of its disparate healthcare systems, enabling healthcare providers to access comprehensive patient information in real-time, resulting in improved clinical decision-making and enhanced patient outcomes.
- **Aerospace and Defense Case Study:** A leading defense contractor utilized CORBA for mission-critical command and control systems. By leveraging CORBA's robust middleware platform, the contractor ensured the reliability and real-time responsiveness of its command and control systems, enabling efficient coordination of military operations and enhancing situational awareness in complex operational environments.

8. Conclusion

- **Architecture and Key Concepts:** CORBA utilizes an Object Request Broker (ORB) to facilitate communication between distributed objects, with support for Interface Definition Language (IDL), Object Adapter, Object Services, and Portable Object Adapter (POA).
- **Advantages:** CORBA offers interoperability, language/platform independence, reusability, scalability, and integration capabilities,

	making it suitable for various industries and distributed computing environments. • **Challenges and Limitations:** Challenges include the complexity of implementations, performance overhead, security concerns, vendor lock-in, and compatibility issues. • **Use Cases and Applications:** CORBA finds applications in industries such as telecommunications, finance, healthcare, and aerospace, enabling enterprise integration, distributed computing, web services, and grid computing. • **Significance of CORBA:** Despite its challenges, CORBA remains significant in the realm of distributed computing due to its robust middleware platform, interoperability, and support for building scalable and interoperable distributed systems. • **Recommendations:** Organizations considering the adoption of CORBA or similar middleware technologies should carefully evaluate their requirements, consider the complexity of implementation, assess performance overhead, address security concerns, mitigate vendor lock-in, and plan for compatibility with existing systems.
Conclusion:	In conclusion, this case study has provided a comprehensive understanding of CORBA and its workings in distributed computing environments. Through detailed exploration and real-world examples, we've grasped CORBA's robust middleware architecture, emphasizing its advantages in interoperability, scalability, and platform independence. As compared to alternative middleware solutions, CORBA's standardized approach and extensive featureset position it as a reliable choice for building distributed systems, showcasing its enduring relevance and superiority in the ever-evolving landscape of distributed computing technologies.
References:	1. https://en.wikipedia.org/wiki/Object_Management_Group 2. http://www.ois.com/Products/communications-middleware.html 3. http://omniorb.sourceforge.net/ 4. https://en.wikipedia.org/wiki/Dr._Dobb%27s_Journal

Rubrics for Assessment

Timely Sub-mission	Late	Submitted just after deadline	On time Submission
	0 points	*1 points*	*2 points*
Under-stand-ing	Student is confused about the concept	Students has justifiably understood the concept	Students is very clear about the concepts
	0 points	*2 points*	*3 points*
Perfor-mance	Students has not per-formed the Experiment	Student has performed with help	Student has independently performed the experiment
	0 points	*2 points*	*3 points*
Devel-op-ment	Student struggles to write code	Student can write code with the requirement stated	Student can write exceptional code with his own ideas
	0 points	*2 points*	*3 points*
Overall Disci-pline in Lab	No Discipline maintained	Follows Rules	Sincerely performs the practi-cal is always on time
	0 points	*1 points*	*4 points*