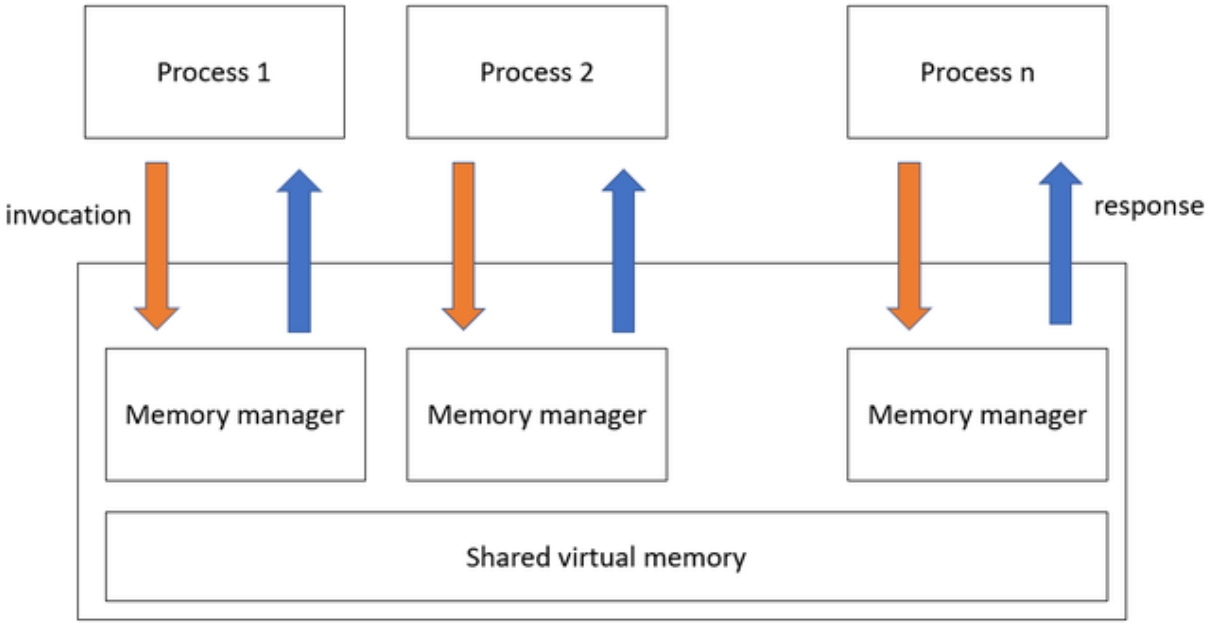
UNIT-5

Distributed Share Memory (DSM) Server

DSM is a mechanism that manages memory across multiple nodes and makes inter-process communications transparent to end-users. The applications will think that they are running on shared memory. DSM is a mechanism of allowing user processes to access shared data without using inter-process communications. In DSM every node has its own memory and provides memory read and write services and it provides consistency protocols. The distributed shared memory (DSM) implements the shared memory model in distributed systems but it doesn't have physical shared memory. All the nodes share the virtual address space provided by the shared memory model. The Data moves between the main memories of different nodes.



Types of Distributed shared memory

On-Chip Memory:

- The data is present in the CPU portion of the chip.
- Memory is directly connected to address lines.
- On-Chip Memory DSM is expensive and complex.

Bus-Based Multiprocessors:

- A set of parallel wires called a bus acts as a connection between CPU and memory.
- accessing of same memory simultaneously by multiple CPUs is prevented by using some algorithms
- Cache memory is used to reduce network traffic.

Ring-Based Multiprocessors:

- There is no global centralized memory present in Ring-based DSM.
- All nodes are connected via a token passing ring.
- In ring-bases DSM a single address line is divided into the shared area.

Advantages of Distributed shared memory

- **Simpler abstraction:** Programmer need not concern about data movement, As the address space is the same it is easier to implement than RPC.
- **Easier portability:** The access protocols used in DSM allow for a natural transition from sequential to distributed systems. DSM programs are portable as they use a common programming interface.
- **Locality of data:** Data moved in large blocks i.e. data near to the current memory location that is being fetched, may be needed future so it will be also fetched.
- **On-demand data movement:** It provided by DSM will eliminate the data exchange phase.
- **Larger memory space:** It provides large virtual memory space, the total memory size is the sum of the memory size of all the nodes, paging activities are reduced.
- **Better Performance:** DSM improve performance and efficiency by speeding up access to data.
- **Flexible communication environment:** They can join and leave DSM system without affecting the others as there is no need for sender and receiver to existing,
- **Process migration simplified:** They all share the address space so one process can easily be moved to a different machine.

Distributed Document-Based System

A Distributed Document-Based System refers to a system architecture where documents (which can include text, images, multimedia, etc.) are stored, managed, and processed across multiple nodes or servers in a distributed network. This approach is often employed to improve scalability, fault tolerance, and performance in comparison to traditional centralized systems. For example .WWW.

World Wide Web (WWW) as Distributed Document Based System

The World Wide Web (WWW) operates as a distributed document-based system through a combination of protocols, technologies, and standards. Here's an overview of how the WWW works in this context:

1. **Hypertext Markup Language (HTML):**
    - Documents on the web are primarily created using HTML. HTML is a markup language that structures content within documents, allowing for the inclusion of text, images, links, and multimedia elements.
2. **Uniform Resource Locators (URLs):**
    - Each document on the web is identified by a unique address called a URL. URLs specify the location of a document and the protocol to use for retrieving it (e.g., HTTP or HTTPS).
3. **HTTP/HTTPS (Hypertext Transfer Protocol/Secure):**
    - The communication between clients (web browsers) and servers is facilitated by the HTTP or HTTPS protocol. HTTP is the standard protocol for transmitting hypertext and multimedia documents on the web.
4. **Web Servers:**
    - Documents are hosted on web servers. These servers respond to client requests by sending the requested documents over the internet.
5. **Web Browsers:**
    - Clients, in the form of web browsers (e.g., Chrome, Firefox, Safari), interpret HTML documents and render them for users. Browsers also handle user interactions such as clicking links and submitting forms.
6. **Hyperlinks:**

- Hyperlinks are a fundamental feature of the web. They allow users to navigate between documents. When a user clicks on a hyperlink, the browser sends a request to the server specified in the URL, and the corresponding document is retrieved and displayed.

7. **Content Delivery Networks (CDNs):**
    - CDNs are used to distribute content across multiple servers globally. This enhances the performance and availability of web documents by caching them closer to users.
8. **DNS (Domain Name System):**
    - DNS translates human-readable domain names into IP addresses that servers can use to identify each other on the internet.
9. **Decentralized Hosting:**
    - While there are centralized components like domain name systems and major server farms, the web's nature is decentralized. Individuals, organizations, and companies can host and publish content on their own servers.
10. **Search Engines:**
    - Search engines index web documents and provide users with a way to discover and navigate the vast amount of information available on the web

Distributed coordination Based System

A distributed coordination-based system is a type of distributed system that focuses on managing and coordinating the activities of multiple components or nodes to achieve a common goal. In distributed systems, where multiple entities (nodes, processes, or services) work together to perform tasks, coordination is essential to ensure consistency, reliability, and efficiency.

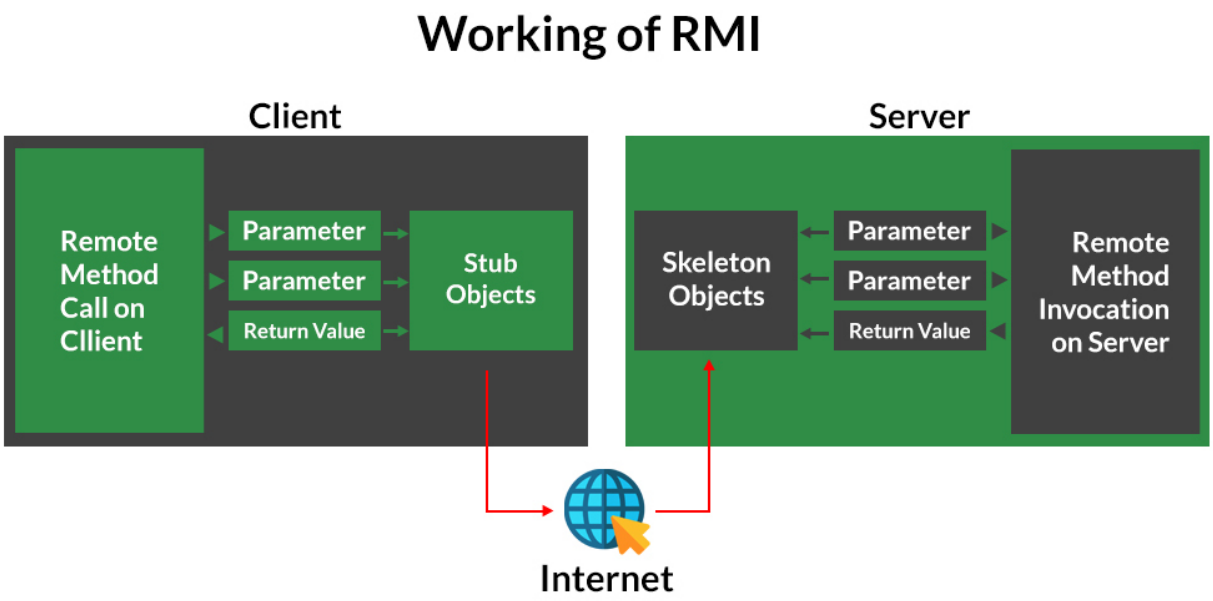
- **Centralized Coordination:** In some distributed systems, coordination is achieved through a centralized coordinator that manages and directs the activities of all nodes.
- **Decentralized Coordination:** In other cases, coordination is decentralized, with nodes collaborating with each other without relying on a central entity.

Java Remote Method Invocation (RMI)

RMI (Remote Method Invocation) is a Java API that allows an object to invoke methods on an object running in another address space, typically on a different machine

Working of RMI

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:



RMI can be categorized into static RMI and dynamic RMI based on the way the remote interface is handled.

1. Static RMI:

- **Compile-Time Binding:**
    - In static RMI, the remote interface is compiled at compile time, and the stub (client-side) and skeleton (server-side) classes are generated during the compilation process.
- **Registry Usage:**
    - A static RMI system typically uses the RMI registry to register and look up remote objects. The registry helps clients discover and connect to remote objects.
- **Code Generation:**
    - The necessary stub and skeleton classes are generated using the **rmic** tool during the compilation phase.

2. Dynamic RMI:

- **Run-Time Binding:**
    - Dynamic RMI, as the name suggests, allows for a more dynamic approach. The stub and skeleton classes are not generated at compile time; instead, they are created at runtime.
- **UnicastRemoteObject and RemoteServer:**
    - Dynamic RMI often utilizes the **UnicastRemoteObject** class and the **RemoteServer** class for creating and exporting remote objects dynamically.
- **No Need for rmic:**
    - In dynamic RMI, there is no need for the **rmic** tool as stubs are generated at runtime.

Key Differences:

1. **Code Generation:**
    - **Static RMI:** Involves code generation at compile time using the **rmic** tool.
    - **Dynamic RMI:** Involves dynamic generation of stubs at runtime, eliminating the need for a separate code generation step.
2. **Registry Usage:**
    - **Static RMI:** Typically relies on the RMI registry for registering and discovering remote objects.
    - **Dynamic RMI:** Can still use the registry, but the dynamic approach allows for more flexibility in object registration and lookup mechanisms.
3. **Flexibility:**
    - **Static RMI:** Offers less flexibility at runtime due to the pre-generated stubs and skeletons.
    - **Dynamic RMI:** Provides more flexibility as stubs are generated dynamically at runtime.
4. **rmic Tool:**
    - **Static RMI:** Requires the use of the **rmic** tool to generate stub and skeleton classes.
    - **Dynamic RMI:** Does not rely on the **rmic** tool as stubs are generated dynamically.

Orbix

Orbix is a CORBA ORB (Object Request Broker) – a software product from Micro Focus (originally from IONA Technologies, then Progress Software, eventually acquired by Micro Focus in December 2012) which helps programmers build distributed applications. Orbix is an implementation of the OMG's (Object Management Group) CORBA Specification.

ORBix is one of the implementations of the CORBA standard. It provides the necessary infrastructure to support the features defined by CORBA. ORBix facilitates the creation, deployment, and communication of distributed objects in accordance with the CORBA specifications.

Key feature of Orbix

- **Interoperability:** ORBix enables interoperability between objects written in different programming languages, running on different platforms.
- **Location Transparency:** ORBix provides location transparency, allowing objects to communicate regardless of their physical location in the network.
- **Object Invocation:** ORBix handles remote method invocations, allowing objects to invoke methods on remote objects seamlessly.
- **Dynamic Invocation:** Some implementations of CORBA, including ORBix, support dynamic invocation of methods on objects, allowing for more flexibility in distributed systems.

System Object Model (SOM)

SOM stands for System Object Model. It is a new model for developing and packaging object-oriented software. SOM offers different benefits to different people:

- To those who build and distribute class libraries: SOM can save the users of the class libraries from recompiling their source code every time there is a new release of the class library.

- To those who would like to adopt object technology but want to use a more familiar language, such as C or COBOL, for their development: SOM offers language bindings that let you implement your classes in a variety of languages.
- To those who want to build distributed applications: the Distributed SOM Framework provides transparent access to objects that are distributed across address spaces on different machines.
- To those who are interested in saving objects: the Persistence SOM Framework allows you to store objects in a persistent store, such as a file system or a relational database, and then restore the objects at a later time.
- To those who are interested in providing workgroup solutions:the Replication SOM Framework allows you to replicate your objects across different address spaces and handles the propagation of updates to all replicated copies of an object.
- To the compiler vendors who would like to supply SOM bindings for their compilers: the Emitter Framework allows you to parse an interface definition so that you can use the returned information to produce your own language emitters, without having to write your own IDL compiler

Need for SOM:

SOM solves this problem by providing a language-neutral object model. Using SOM, classes can be developed in one language, and used in another language. SOM supports the common object-oriented concepts, such as classes, abstraction, and inheritance. This makes it possible for programmers who use procedural languages to also design and implement in an object-oriented fashion.

SOM also allows you to distribute class libraries as dynamic link libraries, without requiring recompilation of application source code. For libraries that are developed in procedure-based language, maintaining upward binary compatibility is normally not a problem. However, for libraries that are developed in object oriented languages such as C++, structural changes in a class interface, such as when new methods are added, will require recompilation of client programs. This can present a serious problem, because recompilation might not be possible in cases when the source code is not available.

SOM Architecture

The Language Neutral Object Interface Definition defines the interface to an object. An interface contains only the information that the client of the object must know in order to use the object. The internal details, or the implementation of the object, are not included in the interface. The interface is described in a formal language called the Interface Definition Language (IDL), which is independent of other programming languages.

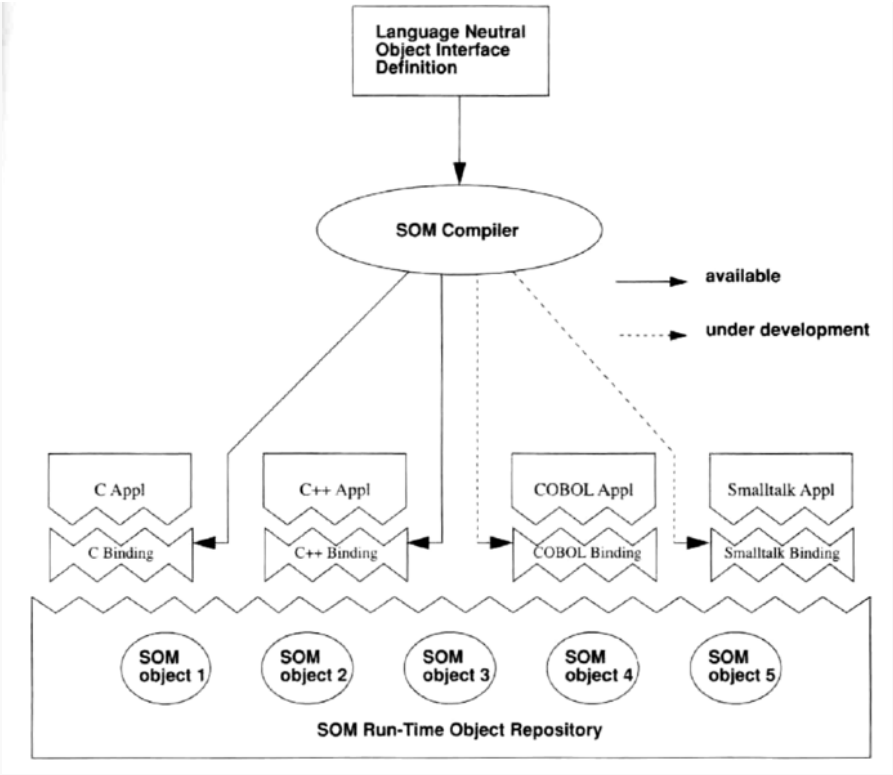


Figure 1.1 The SOM architecture

The SOM compiler compiles the interface definition to produce language-specific bindings. A language binding maps an IDL concept to a specific language construct. For example, an IDL inheritance specification might be mapped directly onto an inheritance construct in a language that supports inheritance, or to a procedure declaration in a language that has no notion of inheritance. The boundary between the language specific bindings to the SOM objects indicates that a client can use any of the supported languages to invoke a SOM object, regardless of the language in which the SOM object is implemented. The protocol between the language bindings and the SOM objects is based on simple procedure calls. For each language binding, the SOM compiler generates the appropriate application programming interfaces (APIs) to the SOM objects.

The SOM Run-Time Object Repository provides functions for creating objects and invoking methods on the objects. It also maintains a registry of all classes in the current process, and assists in the dynamic loading and unloading of classes