//////////////1. Explain Lifecycle of Beans and types of enterprise beans

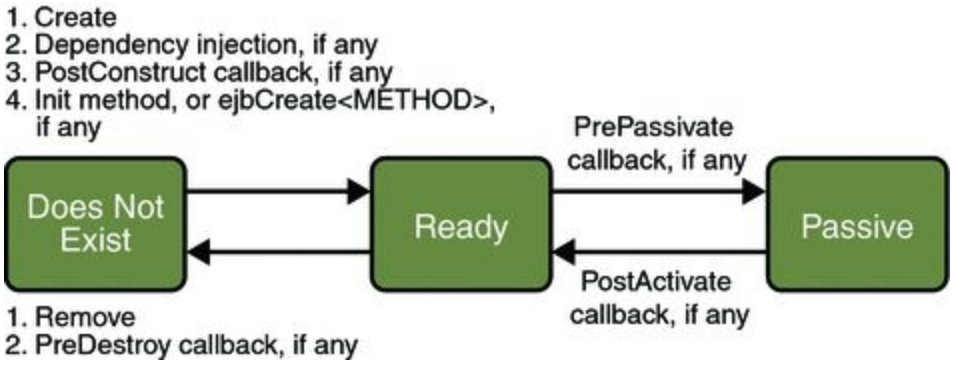
Ans. The Life Cycles of Enterprise Beans

An enterprise bean goes through various stages during its lifetime, or life cycle. Each type of enterprise bean (stateful session, stateless session, or message-driven) has a different life cycle.
The descriptions that follow refer to methods that are explained along with the code examples in the next two chapters. If you are new to enterprise beans, you should skip this section and run the code examples first.

The Life Cycle of a Stateful Session Bean

Figure 20-3 illustrates the stages that a session bean passes through during its lifetime. The client initiates the life cycle by obtaining a reference to a stateful session bean. The container performs any dependency injection and then invokes the method annotated with `@PostConstruct`, if any. The bean is now ready to have its business methods invoked by the client.

Figure 20-3 Life Cycle of a Stateful Session Bean

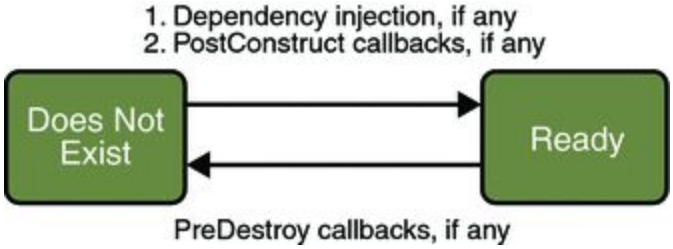


While in the ready stage, the EJB container may decide to deactivate, or **passivate**, the bean by moving it from memory to secondary storage. (Typically, the EJB container uses a least-recently-used algorithm to select a bean for passivation.) The EJB container invokes the method annotated `@PrePassivate`, if any, immediately before passivating it. If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, calls the method annotated `@PostActivate`, if any, and then moves it to the ready stage.
At the end of the life cycle, the client invokes a method annotated `@Remove`, and the EJB container calls the method annotated `@PreDestroy`, if any. The bean's instance is then ready for garbage collection.
Your code controls the invocation of only one life-cycle method: the method annotated `@Remove`. All other methods in Figure 20-3 are invoked by the EJB container. See Chapter 34, Resource Connections for more information.

The Life Cycle of a Stateless Session Bean

Because a stateless session bean is never passivated, its life cycle has only two stages: nonexistent and ready for the invocation of business methods. Figure 20-4illustrates the stages of a stateless session bean.

Figure 20-4 Life Cycle of a Stateless Session Bean

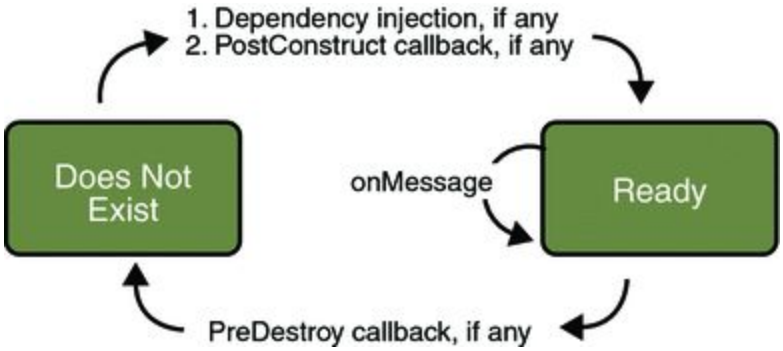


The client initiates the life cycle by obtaining a reference to a stateless session bean. The container performs any dependency injection and then invokes the method annotated `@PostConstruct`, if any. The bean is now ready to have its business methods invoked by the client.
At the end of the life cycle, the EJB container calls the method annotated `@PreDestroy`, if any. The bean's instance is then ready for garbage collection.

The Life Cycle of a Message-Driven Bean

Figure 20-5 illustrates the stages in the life cycle of a message-driven bean.

Figure 20-5 Life Cycle of a Message-Driven Bean



The EJB container usually creates a pool of message-driven bean instances. For each instance, the EJB container performs these tasks:

1. If the message-driven bean uses dependency injection, the container injects these references before instantiating the instance.
2. The container calls the method annotated `@PostConstruct`, if any.

Like a stateless session bean, a message-driven bean is never passivated, and it has only two states: nonexistent and ready to receive messages.

At the end of the life cycle, the container calls the method annotated `@PreDestroy`, if any. The bean's instance is then ready for garbage collection.

1. Session Bean

A Session Bean is an Enterprise Bean that is generated for each session from the client and expires when the client exits. The lifecycle of the Session Bean does not exceed the range from the beginning until the end of the usage of the system by the user. Session Beans are classified into Stateless Session Beans, Stateful Session Beans, and Singleton Session Beans.

a. Stateless Session Bean

This is a model in which the state of the session is not managed. Each session from the client needs to be concluded in one invocation of the Bean business method.

B. Stateful Session Bean

This is a model in which the state of the session is managed. The EJB container manages the state of the session. Even when one session from the client invokes

multiple EJB business methods, the state of each session is saved in between the invocation of the business methods.

C. Singleton Session Bean

This is a model in which the state of the session is shared among multiple clients. One instance which is shared among all the sessions is created for an application. You must determine the Bean lifecycle according to the application.

2. Entity Bean

An Entity Bean expresses the entity, and as a prerequisite, must be stored (persisted) in the database. As a result, even when the client exits, the state of the Entity Bean continues to exist in the database. The lifecycle of this Enterprise Bean is longer as compared to that of a Session Bean. The following two management models are defined in the EJB specifications:

a. BMP(Bean Managed Persistence)

This is a model for managing the data persistence of Enterprise Bean business methods. The developer of the Enterprise Bean must implement processes such as connecting to the database, assembling and executing SQL statements.

B. CMP(Container Managed Persistence)

This is a model in which the EJB container manages the data persistence. The EJB container executes processes, such as connecting to the database and storing data, therefore, these processes need not be executed by the business methods of the Enterprise Bean. Use the method provided by the EJB container to define the mapping of the Enterprise Bean data and the tables and columns of the database in which the data is to be stored. At the same time, define the connection information, such as the host name and port number of the database to which you will connect, in a resource adapter or a data source. The EJB container references this definition

information, assembles the SQL statements, and references and stores the data in the tables of the database to which you will connect.

3. Message driven bean

A Message-driven Bean is a bean that integrates with JMS. The EJB container invokes a Bean, when a JMS message is received from the JMS Destination. Unlike a Session Bean or an Entity Bean, since the Message-driven Bean does not have a home interface and a component interface, it cannot be invoked directly from the client.

2. Explain implementation of sequential consistency model with non replicating migrating blocks strategy.
Ans.

Definition of Sequential Consistency:

The result of any execution is the same as if the (read and write) operations by all processes on the data-store were executed in the same sequential order and the operations of each individual process appear in this sequence in the order specified by its program.

Different Strategies are:

- Non-Replicated, Non-Migrating Blocks(NRNMBs)
- Non-Replicated ,Migrating blocks (NRMBs)
- Replicated, Migrating blocks(RMBs)
- Replicated ,Non-Migrating blocks(RNMBs)

Non-Replicated ,Migrating blocks (NRMBs):

1. In this strategy each block of the shared memory has a single copy in the entire system.However,each access to a block causes the block to migrate from its current node to the node from where it is accessed.
2. When a block is migrated away,it is removed from any local address space it has been mapped into.Notice that in this strategy only the processes executing on one node can read or write a given data item at any one time.Therefore the method ensures sequential consistency.

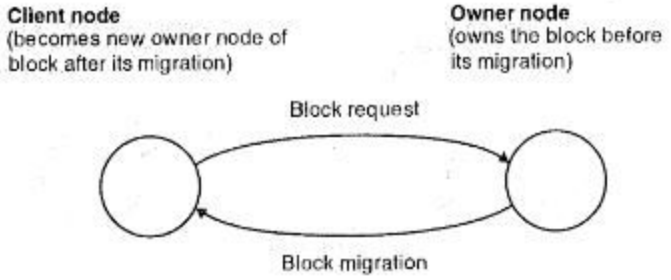
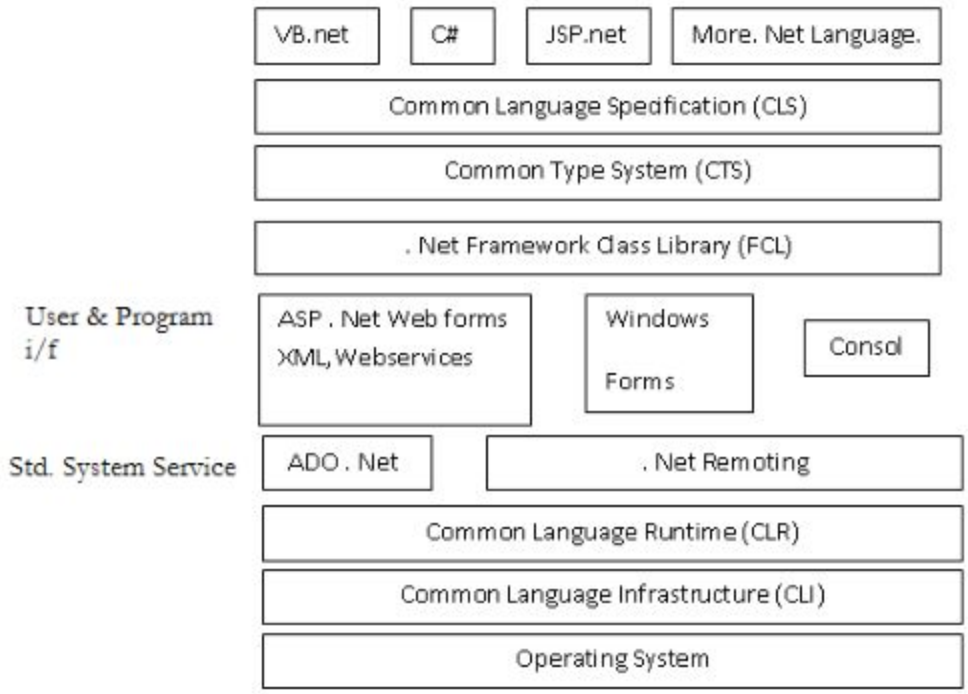


Fig : Nonreplicated,migrating blocks(NRNB) strategy

The method has the following advantages:

1. No communication cost are incurred when a process accesses the data currently held locally.
2. It allows the applications to take advantage of data access locality.If the application exhibits high locality of reference, the cost of data migration is amortized over multiple accesses

3. Describe .NET architecture with neat labeled diagram?
Ans.



the component of .Net framework are describes as follows:

Common Language Runtime(CLR):

CLR monitors the execution of .NET applications and provides essential devices.

It manages code at execution time.

It provide core services like remote communication memory management and thread management.

CLR is equivalent to JVM.

Standard system services:

standard system services like ADO .Net and XML are made universally available and standardized across languages by bringing them under control of .Net framework.

User and program Interface

It includes windows forms that provide powerful user Interface(UT) for web.

.Net Framework class library(FCL)

I(t is a set of managed classes that provide access to system services .

It is comprehensive and object oriented code of reusable types that can be used to develop UI or web applications.

file i/p, o/p's sockets database access remoting are few services of FCL.

Common types system(CTS)

CTS prevents miscasting

It specifies rules related to data types that languages must follow.

CLS is a subset of CTS that allows different languages to interoperate.

CTS is a data log of .Net types.

Common language specification(CLS)

it is set of specifications or guidelines defining a .Net languages.

CLS is a subset objects, CLS defines common types of managed languages.

4. Explain Election Algorithm
Ans

5.Write a short note on Data-Centric Consistency Models.
Ans

Consistency Model

- A consistency model is contract between a distributed data store and processes, in which the processes agree to obey certain rules in contrast the store promises to work correctly.
- A consistency model basically refers to the degree of consistency that should be maintained for the shared memory data.
- If a system supports the stronger consistency model, then the weaker consistency model is automatically supported but the converse is not true.
- The types of consistency models are Data-Centric and client centric consistency models.

1.Data-Centric Consistency Models

A data store may be physically distributed across multiple machines. Each process that can access data from the store is assumed to have a local or nearby copy available of the entire store.

i.Strict Consistency model

- Any read on a data item X returns a value corresponding to the result of the most recent write on X
- This is the strongest form of memory coherence which has the most stringent consistency requirement.
- Strict consistency is the ideal model but it is impossible to implement in a distributed system. It is based on absolute global time or a global agreement on commitment of changes.

ii.Sequential Consistency

- Sequential consistency is an important data-centric consistency model which is a slightly weaker consistency model than strict consistency.
- A data store is said to be sequentially consistent if the result of any execution is the same as if the (read and write) operations by all processes on the data store

were executed in some sequential order and the operations of each individual process should appear in this sequence in a specified order.

- Example: Assume three operations read(R1), write(W1), read(R2) performed in an order on a memory address. Then (R1,W1,R2),(R1,R2,W1),(W1,R1,R2)(R2,W1,R1) are acceptable provided all processes see the same ordering.

iii.Linearizability

- It that is weaker than strict consistency, but stronger than sequential consistency.
- A data store is said to be linearizable when each operation is timestamped and the result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order
- The operations of each individual process appear in sequence order specified by its program.
- If tsOP1(x)< tsOP2(y), then operation OP1(x) should precede OP2(y) in this sequence.

iv.Causal Consistency

- It is a weaker model than sequential consistency.
- In Casual Consistency all processes see only those memory reference operations in the correct order that are potentially causally related.
- Memory reference operations which are not related may be seen by different processes in different order.
- A memory reference operation is said to be casually related to another memory reference operation if the first operation is influenced by the second operation.
- If a write(w2) operation is casually related to another write (w1) the acceptable order is (w1, w2).

v.FIFO Consistency

- It is weaker than causal consistency.
- This model ensures that all write operations performed by a single process are seen by all other processes in the order in which they were performed like a single process in a pipeline.
- This model is simple and easy to implement having good performance because processes are ready in the pipeline.
- Implementation is done by sequencing write operations performed at each node independently of the operations performed on other nodes.

- Example: If (w11) and (w12) are write operations performed by p1 in that order and (w21),(w22) by p2. A process p3 can see them as [(w11,w12),(w21,w2)] while p4 can view them as [(w21,w2),(w11,w12)].

vi.Weak consistency

- The basic idea behind the weak consistency model is enforcing consistency on a group of memory reference operations rather than individual operations.
- A Distributed Shared Memory system that supports the weak consistency model uses a special variable called a synchronization variable which is used to synchronize memory.
- When a process accesses a synchronization variable, the entire memory is synchronized by making visible the changes made to the memory to all other processes.

vii.Release Consistency

- Release consistency model tells whether a process is entering or exiting from a critical section so that the system performs either of the operations when a synchronization variable is accessed by a process.
- Two synchronization variables acquire and release are used instead of single synchronization variable. Acquire is used when process enters critical section and release is when it exits a critical section.
- Release consistency can be viewed as synchronization mechanism based on barriers instead of critical sections.

viii.Entry Consistency

- In entry consistency every shared data item is associated with a synchronization variable.
- In order to access consistent data, each synchronization variable must be explicitly acquired.
- Release consistency affects all shared data but entry consistency affects only those shared data associated with a synchronization variable.

6.Explain Client Centric Consistency Models

Ans

2.Client-Centric Consistency Models

- Client-centric consistency models aim at providing a system wide view on a data store.

- This model concentrates on consistency from the perspective of a single mobile client.
- Client-centric consistency models are generally used for applications that lack simultaneous updates were most operations involve reading data.

i.Eventual Consistency

- In Systems that tolerate high degree of inconsistency, if no updates take place for a long time all replicas will gradually and eventually become consistent. This form of consistency is called eventual consistency.
- Eventual consistency only requires those updates that guarantee propagation to all replicas.
- Eventual consistent data stores work fine as long as clients always access the same replica.
- Write conflicts are often relatively easy to solve when assuming that only a small group of processes can perform updates. Eventual consistency is therefore often cheap to implement.

ii.Monotonic Reads Consistency

- A data store is said to provide monotonic-read consistency if a process reads the value of a data item x, any successive read operation on x by that process will always return that same value or a more recent value.
- A process has seen a value of x at time t, it will never see an older version of x at a later time.
- Example: A user can read incoming mail while moving. Each time the user connects to a different e-mail server, that server fetches all the updates from the server that the user previously visited. Monotonic Reads guarantees that the user sees all updates, no matter from which server the automatic reading takes place.

iii.Monotonic Writes

- A data store is said to be monotonic-write consistent if a write operation by a process on a data item x is completed before any successive write operation on X by the same process.
- A write operation on a copy of data item x is performed only if that copy has been brought up to date by means of any preceding write operations, which may have taken place on other copies of x.
- Example: Monotonic-write consistency guarantees that if an update is performed on a copy of Server S, all preceding updates will be performed first. The resulting

server will then indeed become the most recent version and will include all updates that have led to previous versions of the server.

iv.Read Your Writes

- A data store is said to provide read-your-writes consistency if the effect of a write operation by a process on data item x will always be a successive read operation on x by the same process.
- A write operation is always completed before a successive read operation by the same process no matter where that read operation takes place.
- Example: Updating a Web page and guaranteeing that the Web browser shows the newest version instead of its cached copy.

v.Writes Follow Reads

- A data store is said to provide writes-follow-reads consistency if a process has write operation on a data item x following a previous read operation on x then it is guaranteed to take place on the same or a more recent value of x that was read.
- Any successive write operation by a process on a data item x will be performed on a copy of x that is up to date with the value most recently read by that process.
- Example: Suppose a user first reads an article A then posts a response B. By requiring writes-follow-reads consistency, B will be written to any copy only after A has been written.

7.Explain lamport clock.

Ans

In a distributed system, it is not possible in practice to synchronize time across entities (typically thought of as processes) within the system; hence, the entities can use the concept of a logical clock based on the events through which they communicate.

- If two entities do not exchange any messages, then they probably do not need to share a common clock; events occurring on those entities are termed as concurrent events.
- Among the processes on the same local machine we can order the events based on the local clock of the system.
- When two entities communicate by message passing, then the send event is said to *happen-before* the receive event, and the logical order can be established among the events.
- A distributed system is said to have partial order if we can have a partial order relationship among the events in the system. If 'totality', i.e., causal relationship among all events in the system, can be established, then the system is said to have total order.

- A single entity cannot have two events occur simultaneously. If the system has total order we can determine the order among all events in the system. If the system has partial order between processes, which is the type of order Lamport's logical clock provides, then we can only tell the ordering between entities that interact. Lamport addressed ordering two events with the same timestamp (or counter): "To break ties, we use any arbitrary total ordering < of the processes.Thus two timestamps or counters may be the same within a distributed system, but in applying the logical clocks algorithm events that occur will always maintain at least a strict partial ordering.
- Lamport's logical clocks lead to a situation where all events in a distributed system are totally ordered. That is, if a->b, then we can say a actually happened before b.
- Unfortunately, with Lamport's clocks, nothing can be said about the actual time of a and b. If the logical clock says a->b, that does not mean in reality that a actually happened before b in terms of real time.
- The problem with Lamport clocks is that they do not capture causality.
- If we know that we cannot say which action initiated c.
- This kind of information can be important when trying to replay events in a distributed system (such as when trying to recover after a crash).
- The theory goes that if one node goes down, if we know the causal relationships between messages, then we can replay those messages and respect the causal relationship to get that node back up to the state it needs to be in.

8.Write short note on Physical and Logical Clock.

Physical Clocks

- Computer Timer: an integrated circuit that contains a precisely machined quartz crystal. When kept under tension the quartz crystal oscillates at a well-defined frequency.
- Clock Tick: after a predefined number of oscillations, the timer will generate a clock tick. This clock tick generates a hardware interrupt that causes the computer's operating system to enter a special routine in which it can update the software clock and run the process scheduler.
- **This system is fairly reliable on one system. With the timer we can define:**
    - simultaneous: all actions that happen between clock ticks
    - before: an operation that happens in a previous clock tick
    - after: an operation that happens in a subsequent clock tick

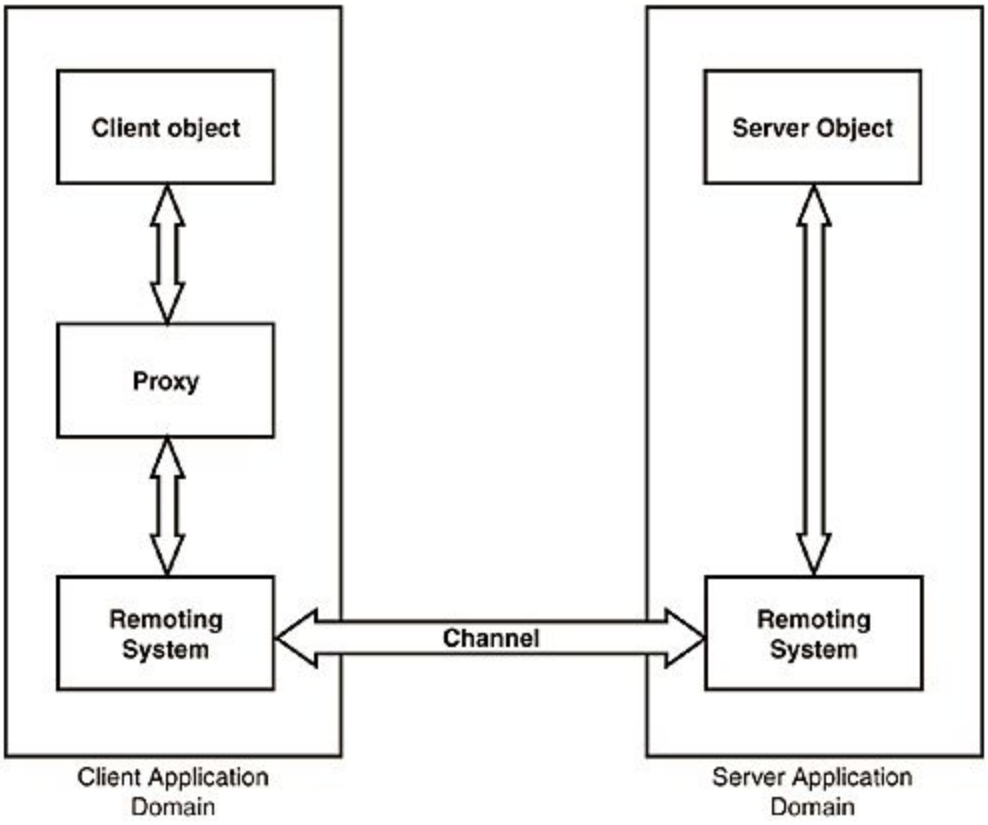
Physical Clocks - Multiple Systems

- Unfortunately, it is impossible for each machined quartz crystal in every computer timer to be exactly the same. These differences create clock skew.
- For example, if a timer interrupts 60 times per second, it should generate 216,000 ticks per hour.
- In practice, the real number of ticks is typically between 215,998 and 216,002 per hour. This means that we aren't actually getting precisely 60 ticks per second.
- We can say that a timer is within specification if there is some constant p such that:

$$1-p<=dCdT<=1+p1-p<=dCdT<=1+p$$

- The constant p is the maximum drift rate of the timer.
- On any two given computers, the drift rate will likely differ.
- To solve this problem, clock synchronization algorithms are necessary.

9.Explain .NET Remoting.
Ans.



.NET Remoting is a mechanism that allows objects to interact with each other across application domains, whether application components are all on one computer or spread out across the entire world. Remoting was designed in such a

way that it hides the most difficult aspects like managing connections, marshaling data, and reading and writing XML and SOAP.

.NET Remoting is a mechanism for communicating between objects which are not in the same process. It is a generic system for different applications to communicate with one another. .NET objects are exposed to remote processes, thus allowing inter process communication. The applications can be located on the same computer, different computers on the same network, or on computers across separate networks.
Microsoft .NET Remoting provides a framework that allows objects to interact with each other across application domains. Remoting was designed in such a way that it hides the most difficult aspects like managing connections, marshaling data, and reading and writing XML and SOAP. The framework provides a number of services, including object activation and object lifetime support, as well as communication channels which are responsible for transporting messages to and from remote applications.
Both application domains communicate with each other by following these steps:

1. When a client object requests an instance of the server object, the remoting system at the client side instead creates a proxy of the server object. The proxy object lives at the client but behaves just like the remote object; this leaves the client with the impression that the server object is in the client's process.
2. When the client object calls a method on the server object, the proxy passes the call information to the remoting system on the client. This remoting system in turn sends the call over the channel to the remoting system on the server.
3. The remoting system on the server receives the call information and, on the basis of it, invokes the method on the actual object on the server (creating the object if necessary).
4. The remoting system on the server collects the result of the method invocation and passes it through the channel to the remoting system on the client.
5. The remoting system at the client receives the server's response and returns the results to the client object through the proxy.

The process of packaging and sending method calls among objects, across application boundaries, via serialization and deserialization, as shown in the preceding steps, is also known as *marshaling*.

10. Explain four different distributed deadlock detection algorithms.
Ans
Distributed deadlock detection algorithms can be divided into four classes:

1. Path-Pushing Algorithms

In path-pushing algorithms, distributed deadlocks are detected by maintaining an explicit global WFG. The basic idea is to build a global WFG for each site of the distributed system. In this class of algorithms, at each site whenever deadlock computation is performed, it sends its local WFG to all the neighboring sites

2. Edge-Chasing Algorithms

In an edge-chasing algorithm, the presence of a cycle in a distributed graph structure is be verified by propagating special messages called probes, along the edges of the graph. These probe messages are different than the request and reply messages. The formation of cycle can be deleted by a site if it receives the matching probe sent by it previously.

3. Diffusing Computations Based Algorithms

In diffusion computation based distributed deadlock detection algorithms, deadlock detection computation is diffused through the WFG of the system. These algorithms make use of echo algorithms to detect deadlocks. This computation is superimposed on the underlying distributed computation. If this computation terminates, the initiator declares a deadlock global state detection.

4. Global State Detection Based Algorithms

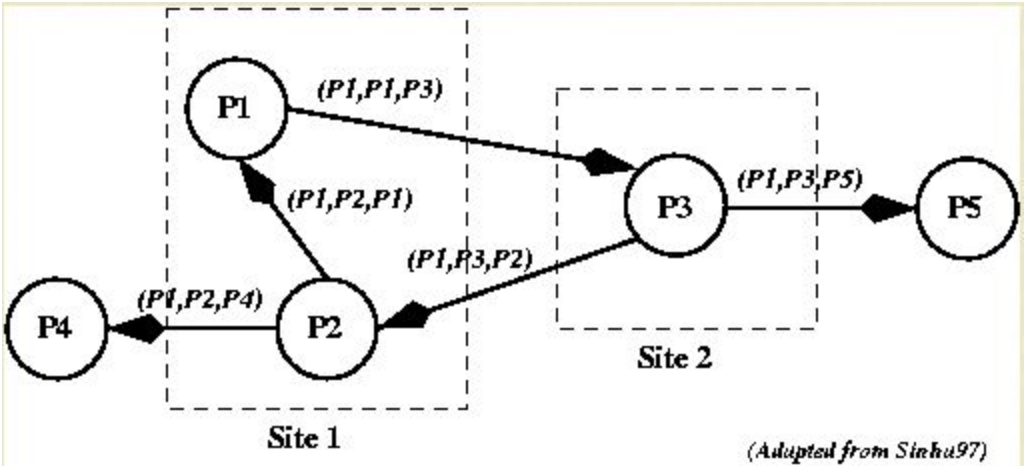
Global state detection based deadlock detection algorithms exploit the following facts:

1 A consistent snapshot of a distributed system can be obtained without freezing the underlying computation and

2 If a stable property holds in the system before the snapshot collection is initiated, this property will still hold in the snapshot.

Probe Based distributed Deadlock Detection Algorithm:

- This is considered an edge-chasing, probe-based algorithm. It is also considered one of the best deadlock detection algorithms for distributed systems.

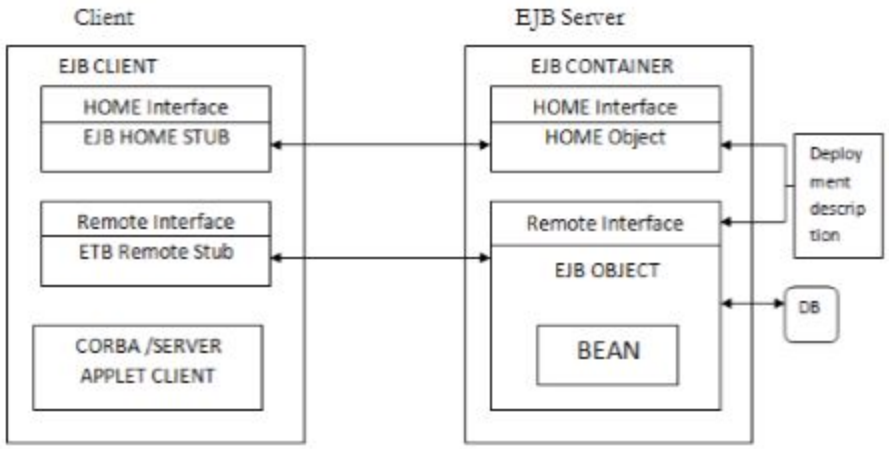
- If a process makes a request for a resource which fails or times out, the process generates a probe message and sends it to each of the processes holding one or more of its requested resources.
- Each probe message contains the following information:
    1. the id of the process that is blocked (the one that initiates the probe message);
    2. the id of the process is sending this particular version of the probe message; and
    3. the id of the process that should receive this probe message.
- When a process receives a probe message, it checks to see if it is also waiting for resources.
- If not, it is currently using the needed resource and will eventually finish and release the resource.
- If it is waiting for resources, it passes on the probe message to all processes it knows to be holding resources it has itself requested.
- The process first modifies the probe message, changing the sender and receiver ids.
- If a process receives a probe message that it recognizes as having initiated, it knows there is a cycle in the system and thus, deadlock.
- The following example is based on the same data used in the Silberschatz-Galvin algorithm example. In this case P1 initiates the probe message, so that all the messages shown have P1 as the initiator. When the probe message is received by process P3, it modifies it and sends it to two more processes. Eventually, the probe message returns to process P1. Deadlock!
- 

ADD COMMENT • link

11. Explain components of ejb framework

Ans

- JB (Enterprise Java Beans) is a server-side component that executes specific business logic.
- It enables development and deployment of component based, robust, highly scalable & transactional business application.



1) EJB Server :

- It provides execution environment for server components.
- It hosts EJB containers.

2) EJB Container:

- It serves as interface EJB and Outside World.

Container hosts following components:

- EJB Object: A client invokes beans instance through EJB Object. It is called Request Interceptor.
- Remote Interface: It duplicates machines of bean class.
- Home Object: It is the object factory bean where client can create or destroy object.
- Home Interface: It provides machine for creating and destroying, finding bean object.
- Local Interface: It is used instead of EJB objects for faster access.
- The type of container depends on beans they contains i.e either transient or persistent beans.

3) Enterprise Beans:

- They are EJB Components that encapsulates business functionalities in an application.
- Container provides security to enterprise beans.
- Types of Enterprise Beans:
    - Session Beans(Synchronous)
    - Message Driven(Asynchronous)
    - Entity Beans (Data/Records in DB).

4) EJB Client:

- It makes use of EJB beans to perform the operations.
- Client locates the container using JNDI (Java Naming & Directory Interface)
- Then the client invokes JB Beans machines through container.

5) Deployment Descriptor:

- It lists the bean properties and elements like:
    - JNDI Name for Bean.
    - Home and Remote Interface.
    - Bean Implementation Class.
    - Environment Variables.
    - Access Rules or Rights.
    - Beans Management, Etc.

12. Comparison of NOS and DOS
Ans.
User awareness :

NOS: Users are aware of multiplicity of machines.

DOS: Users are not aware of multiplicity of machines.

Resource Access:

NOS: Access to resources of various machines is done explicitly by remote logging into the appropriate remote machine or transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism.

DOS: Access to remote resources similar to access to local resources.

Computation and Data Migration:

NOS: transfer the the data, to and from the remote server and only the server performs the all or most of the computation.

DOS: transfer the computation, rather than the data, across the system.

Process Migration:

NOS: execute an entire process, or parts of it, at the remote server. DOS: execute an entire process, or parts of it, at different sites.

Data Access:

NOS: run process remotely, and needs to transfer all the data to the server for processing.

DOS: run process remotely, rather than transfer all data locally.

Architecture Model:

NOS: Employs a client-server model

DOS: Employs a master-slave model.

13. explain corba components.
Ans

- CORBA is a standard for Distributed Object Middleware defined by OMG (Object Management Group).
- Conformance to the CORBA's specifications enables development of heterogeneous application across all major hardware platforms and operating system.
- OMG provides specification to encapsulate application data and business logic within objects called Distributed Components.

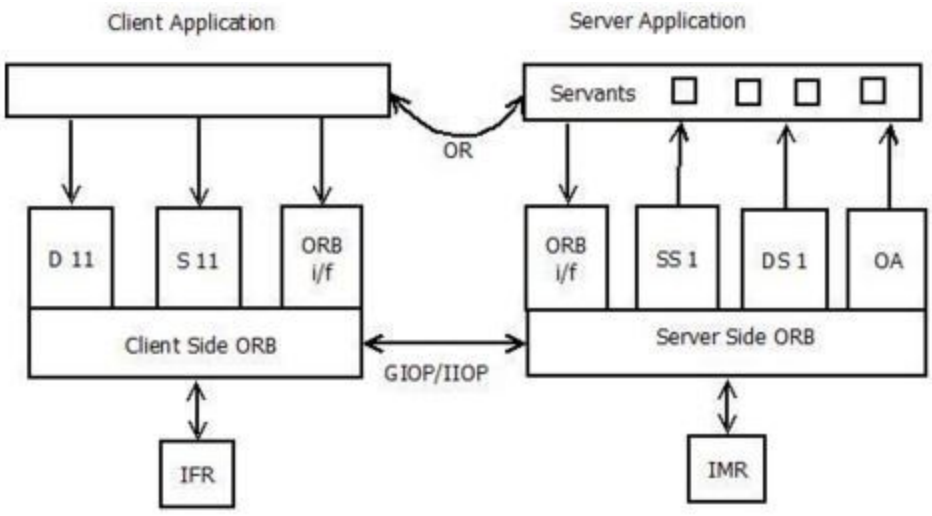


Figure: Common Object Request Broker Architecture

- The Major Components of CORBA are as follows :
- ORB(Object Request Broker):
- 1)It provides the communication and activation infrastructure for distributed object applications.
- 2)To make a request, client specifies the target object by using an Object Reference (OR).
- 3) When a CORBA Object is created, an OR is also created.
- 4) It is unique, immutable and opaque.
- 5) ORB Facilities communication transparency in terms of :
    - **Object Location:** The client has no knowledge of whether the target object is in process or out-process on same machine or different machine.
    - ii) **Object Implementation:** Here, the client does not know the language platform or hardware in which object is implemented.
    - iii) **Object Execution State:** Here, the client does not know whether the object is active or in-active. ORB transparently activates object when required by client.
    - iv) **Object Communication Mechanism:** ORB can use memory, RPC & TCP/IP for communication.
- **OMG Interface Definition Language (OMG-IDL):**
- An object i/ifs defines the requests that can be made by the object.
- It is a declarative language which forces clear separation of i/ifs from object implementation.
- **Stubs & Skeletons:**

- A client side stub is a mechanism that effectively creates an issue request on behalf of a client.
- While server side skeleton is a mechanism that delivers request to the CORBA object implementation.
- As they are i/f-specific dispatching through stubs and skeletons is called Static Inovocation.

IDL Compiler:

- It generates static stubs and skeletons SI & SSI (Static Invocation/Skeletons from ilf definition)

It allows object invocation through SSI & SII.

IFR (Interface Repository):

- It allows system to be accessed and written programmatically at runtime so it is used in dynamic object invocation.

DII (Dynamic I/f Invocation):

- It allows client to generate request at run time.
- DLL allows an ORB to deliver request to servants that have no compile-time knowledge of i/f it accesses.

Object Adapter(QA):

- It serves as a Glue between CORBA Object Implementation and ORB.
- It is an object that add up if of an object to the i/f expected by a caller.

There are two types of QA :

BOA(Basic QA):

BOA defines an object adapter which can be used for most conventional object implementation.

POA (Portable QA)

POA specifications provide portability for CORBA Server code.

Inter-ORB Protocol:

To support interoperability CORBA processes :

- GIOP(General Inter-operable Protocol):

- `It defines common mechanism of communication between ORB's in general forms.`

It describes CDR (Common Character Data Representation) and message formats between ORB's.

- IIOP(Internet Interoperable Protocol)
- `IIOP provides full duplex, connection-oriented communication channel through TCP/IP Protocol.`

It gives client an ability to access that object while hiding details of that object implementation and status.

- ORB if:

To decouple applications from implementations details if is defined.

Thus provides standard operations to initialize and shutdown ORB, convert OR to string and back.

IMR (Implementation Repository)

It contains information that allows an ORB to activate servers on demand.

The IMR maintains table that associates table name with start command.

14. explain ejb roles in ejb and types of bean.
Ans.
EJB specification gives a clear description of each of these roles and their associated responsibilities.
a) Enterprise Bean Provider

- It is the person or group that writes and packages EJBs. It might be a third-party vendor of EJB components, or it might be an information systems programmer who writes EJBs to implement their company's business logic.
- Because the container must provide transaction and network communication support, the enterprise bean provider does not need to be an expert at low-level system programming, networking or transaction processing.

b) Deployer

- The deployer takes an ejb-jar file and installs it into an EJB container.
- The deployer's task begins with the receipt of an ejb-jar file, and ends with the installation of the ejb-jar file in the container.
- This task is typically handled by the System Administrator in MIS world.

c) Application Assembler

- An application assembler builds applications using EJBs classes.
- An MIS programmer may purchase a prepackaged set of EJBs that implement an accounting system.
- Then the programmer might use tools to customize the beans.

d) EJB Server Provider

- An EJB Server Provider provides a server that can contain EJB containers.
- The EJB 1.0 specification does not describe the interface between the server and the container, so the EJB server provider and the EJB container provider will likely be one and the same for any given product.

e) EJB Container Provider

- An EJB container is the world in which an EJB bean lives.
- The container services requests from the EJB, forwards requests from the client to the EJB, and interacts with the EJB server.
- The container provider must provide transaction support, security and persistence to beans.
- It is also responsible for running the beans and for ensuring that the beans are protected from the outside world.

f) System Administrator

- System Administrators are responsible for the day-to-day operation of the EJB server.
- Their responsibilities include keeping security information up-to-date and monitoring the performance of the server.

Later half Q1

15. State difference between Statefull and State less session bean
Ans .

Stateless:

1) Stateless session bean maintains across method and transction

2) The EJB server transparently reuses instances of the Bean to service different clients at the per-method level (access to the session bean is serialized and is 1 client per session bean per method.

3) Used mainly to provide a pool of beans to handle frequent but brief reuests. The EJB server transparently reuses instances of the bean to service different clients.

4) Do not retain client information from one method invocation to the next. So many requir the client tomaintain on the client side which can mean more complex client code.

5) Client passes needed information as parameters to the business methods.6) Performance can be improved due to fewer connections across the network.

Stateful:

1) A stateful session bean holds the client session's state.

2) A stateful session bean is an extesion of the client that creates it.

3) Its fields contain a conversational state on behalf of the session object's client. This state describes the conversation represented by a specific client/session object pair.

4) Its lifetime is controlled by the client.

5) Cannot be shared between clients.