Presented by: Dr. R. BENTRCIA

Department of Computer Science, M’sila University

Outline

- Objectives
- Distributed Objects
  - Organization
  - Types
  - Communication
  - Object Server
- Enterprise Javabeans EJB
  - Definition
  - Motivations
  - Types
- References

Objectives

- To realize the concept of distributed objects.
- To know the General architecture of an EJB server..

Distributed Objects

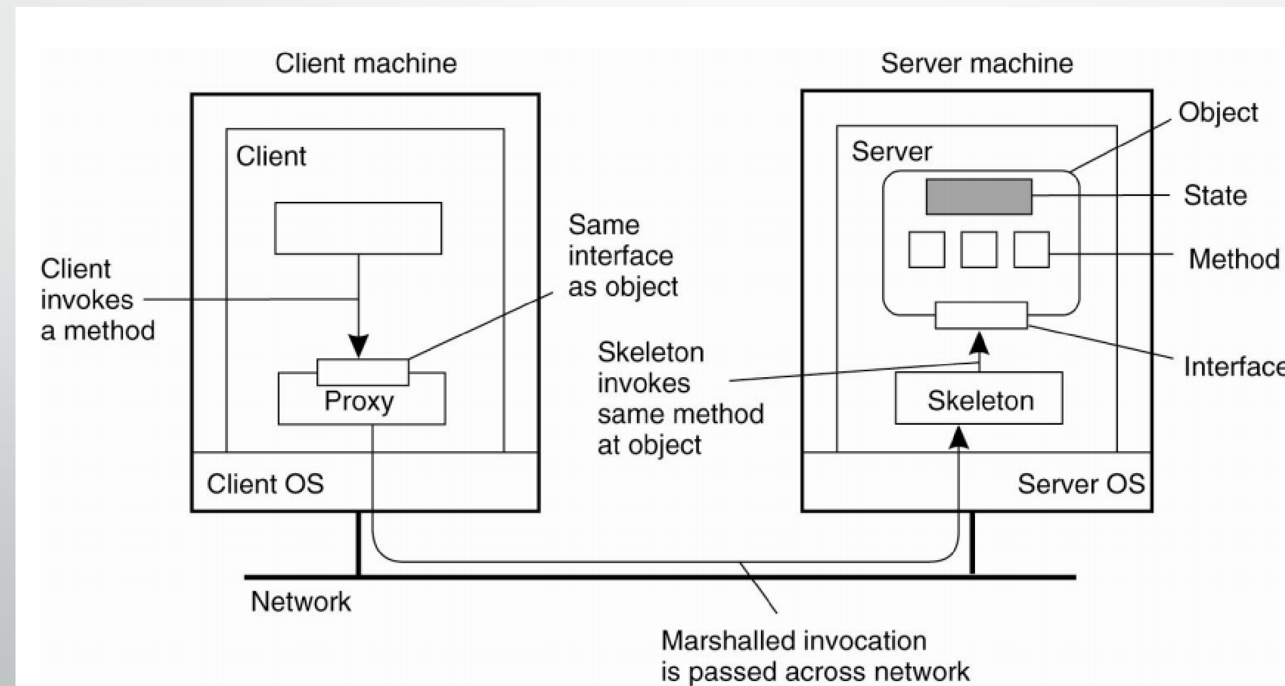
- In distributed object based systems, the notion of an *object* plays a key role in establishing distribution *transparency*.
- In principle, everything is treated as an object and clients are offered services and resources in the form of objects that they can invoke.
- Distributed objects form an important paradigm because it is relatively easy to hide distribution aspects behind an object's interface.

Distributed Objects

- The key feature of an object is that it encapsulates data called the state, and the operations on those data, called the methods. Methods are made available through an interface.
- This separation between interfaces and the objects implementing these interfaces is crucial for distributed systems. A strict separation allows us to place an interface at one machine, while the object itself resides on another machine.
- A characteristic of most distributed objects is that their state is not distributed: it resides at a single machine. Only the interfaces implemented by the object are made available on other machines. Such objects are also referred to as *remote objects.*

Distributed Objects

- Common organization of a remote object with client-side proxy:



Distributed Objects

- **The Client**: When a client binds to a distributed object, an implementation of the object's interface, called a *proxy*, is then loaded into the client's address space.
- The proxy marshals method invocations into messages and unmarshals reply messages to return the result of the method invocation to the client. The actual object resides at a server machine, where it offers the same interface as it does on the client machine. Incoming invocation requests are first passed to a server stub, which unmarshals them to make method invocations at the object's interface at the server.
- **The server** stub is also responsible for marshaling replies and forwarding reply messages to the client side proxy.

Distributed Objects

- Two types of distributed objects are: *Persistent and Transient Objects*.
- *Persistent objects* will survive even if their server is shut down.
- A persistent object continues to exist even if it is currently not contained in the address space of any server process. In other words, a persistent object is not dependent on its current server.
- In practice, this means that the server that is currently managing the persistent object, can store the object's state on secondary storage and then exit. Later, a newly started server can read the object's state from storage into its own address space, and handle invocation requests.

Distributed Objects

- A transient object is an object that exists only as long as the server that is hosting the object exists too.
- *Transient objects* die if their server is shut down.
- Most object-based distributed systems simply support both types.

Distributed Objects

- Distributed systems generally offer the means for a remote client to invoke an object.
- This mechanism is largely based on providing object references.
- Object references can be freely passed between processes (client and server) on different machines, for example as parameters to method invocations.
- By hiding the actual implementation of an object reference, distribution transparency is enhanced.
- When a process holds an object reference, it must first bind to the referenced object before invoking any of its methods.
- Binding results in a proxy being placed in the process's address space, implementing an interface containing the methods the process can invoke.

Distributed Objects

- In many cases, binding is done automatically.
- When an object reference is given, it needs a way to locate the server that manages the actual object, and place a proxy in the client's address space.
- With *implicit binding*, the client is offered a simple mechanism that allows it to directly invoke methods using only a reference to an object.
- In contrast, with *explicit binding*, the client should first call a special function to bind to the object before it can actually invoke its methods.
- Explicit binding generally returns a pointer to a proxy that is then become locally available.

Object Servers

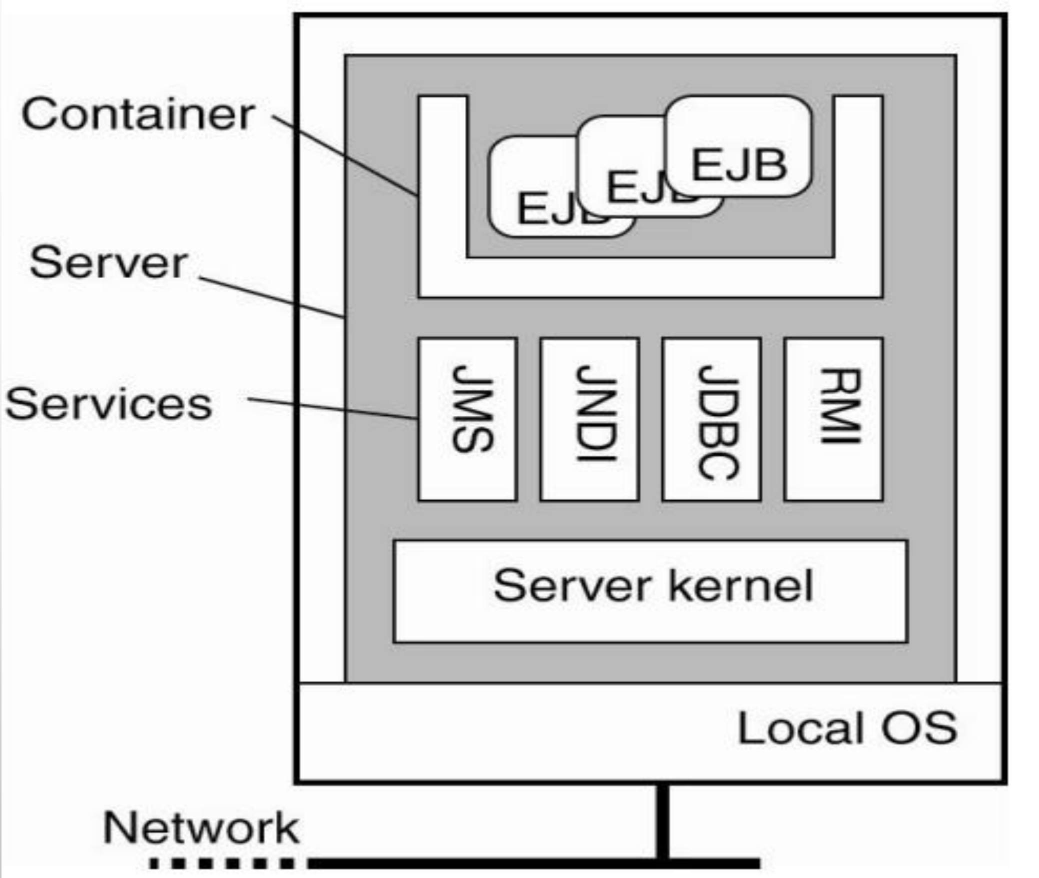
- An object server is a server designed to host distributed objects.
- The important difference between a general object server and other servers is that an object server by itself does not provide a specific service.
- Specific services are implemented by the objects that reside in the server.
- Essentially, the server provides only the means to invoke local objects, based on requests from remote clients.
- It is relatively easy to change services by simply adding and removing objects. An object server thus acts as a place where objects live.

Enterprise JavaBeans EJB

- An EJB is essentially a Java object that is hosted by a special server offering different ways for remote clients to invoke that object.
- EJB is embedded inside a container which effectively provides *interfaces* to underlying services that are implemented by the application server.
- Typical services include those for remote method invocation (RMI), Java Database Connectivity (JDBC), Java Naming Directory Interface (JNDI), and Java Message Service (JMS).

Enterprise JavaBeans EJB

- General architecture of an EJB server:



Motivations

- When to Use Enterprise Beans?
- The application must be *scalable*. To accommodate a growing number of users, you may need to distribute an application’s components across multiple machines. Not only can the enterprise beans of an application run on different machines, but also their location will remain *transparent* to the clients.
- Transactions must ensure data *integrity*. Enterprise beans support transactions, the mechanisms that manage the concurrent access of shared objects.
- The application will have a variety of clients. With only a few lines of code, remote clients can easily locate enterprise beans. These clients can be *thin*, *various, and numerous.*

Types of Enterprise JavaBeans

- There are two basic types of Enterprise JavaBeans: *session* beans and *entity* beans.
- A *session bean* is an EJB instance associated with a single client. Session beans typically are *not persistent* (although they can be), and may or may not participate in transactions. In particular, session objects generally don't survive server crashes.
- One example of a session object might be an EJB living inside a Web server that serves HTML pages to a user on a browser, and tracks that user's path through the site. When the user leaves the site, or after a specified idle time, the session object will be destroyed.

Types of Enterprise JavaBeans

- Session beans can be stateless or stateful.
- Stateless session beans are transient objects that are invoked once, do their work, after which they discard any information they maintain to perform the service they offered to a client.
  - e.g. an SQL query
- Stateful session beans maintain client-related state.
  - Remember client data
  - e.g. what a client clicks and writes on web forms when shopping on Internet.

Types of Enterprise JavaBeans

- An *entity bean* represents information *persistently stored in a database*. Entity beans are associated with database transactions, and may provide data access to multiple users. Since the data that an entity bean represents is persistent, entity beans survive server crashes (this is because when the server comes back online, it can reconstruct the bean from the underlying data).
- An example of an entity bean might be an Employee object for a particular employee in a company's Human Resources database.

Types of Enterprise JavaBeans

- Message-driven beans are used to program objects that should react to incoming messages and be able to send messages.
- They cannot be invoked directly by a client, but rather fit into a publish-subscribe way of communication.

References

- Tanenbaum & Van Steen, Distributed Systems: Principles and Paradigms, 2nd edition, 2007, Prentice-Hall, Inc.