4. Distributed Heterogeneous Applications and CORBA

BY ANKU JAISWAL
ASST. PROF. IOE PULCHOWK CAMPUS

CONTENT

Overview of Heterogeneous System

The distributed system contains many different kinds of hardware and software working together in cooperative fashion to solve problems.

There may be many different representations of data in the system.

This might include different representations for integers, byte streams, floating point numbers, and character sets.

Most of the data can be marshaled from one system to another without losing significance.

There may be many different instructions sets.

An application compiled for one instruction set can not be easily run on a computer with another instruction set unless an instruction set interpreter is provided.

Some components in the distributed system may have different capabilities than other components. Among these might include faster clock cycles, larger memory capacity, bigger disk farms, printers and other peripherals, and different services. Seldom are any two computers exactly alike.

4.1. HETEROGENEITY IN DISTRIBUTED SYSTEM

Heterogeneity refers to the **ability for the system to operate on a variety of different hardware and software components.**

This is achieved through the **implementation of middleware** in the software layer.

Heterogeneity is one of the challenges of a distributed system that **refers to differences in hardware, software, or network configurations among nodes.**

"heterogeneous distributed computing", they mean two or more computers with different hardware being used together to solve a problem.

- The distributed system contains many different kinds of hardware and software **working together in a cooperative fashion** to solve problems.

- There may be many **different representations of data** in the system.

- This might include different representations for integers, byte streams, floating-point numbers, and character sets.

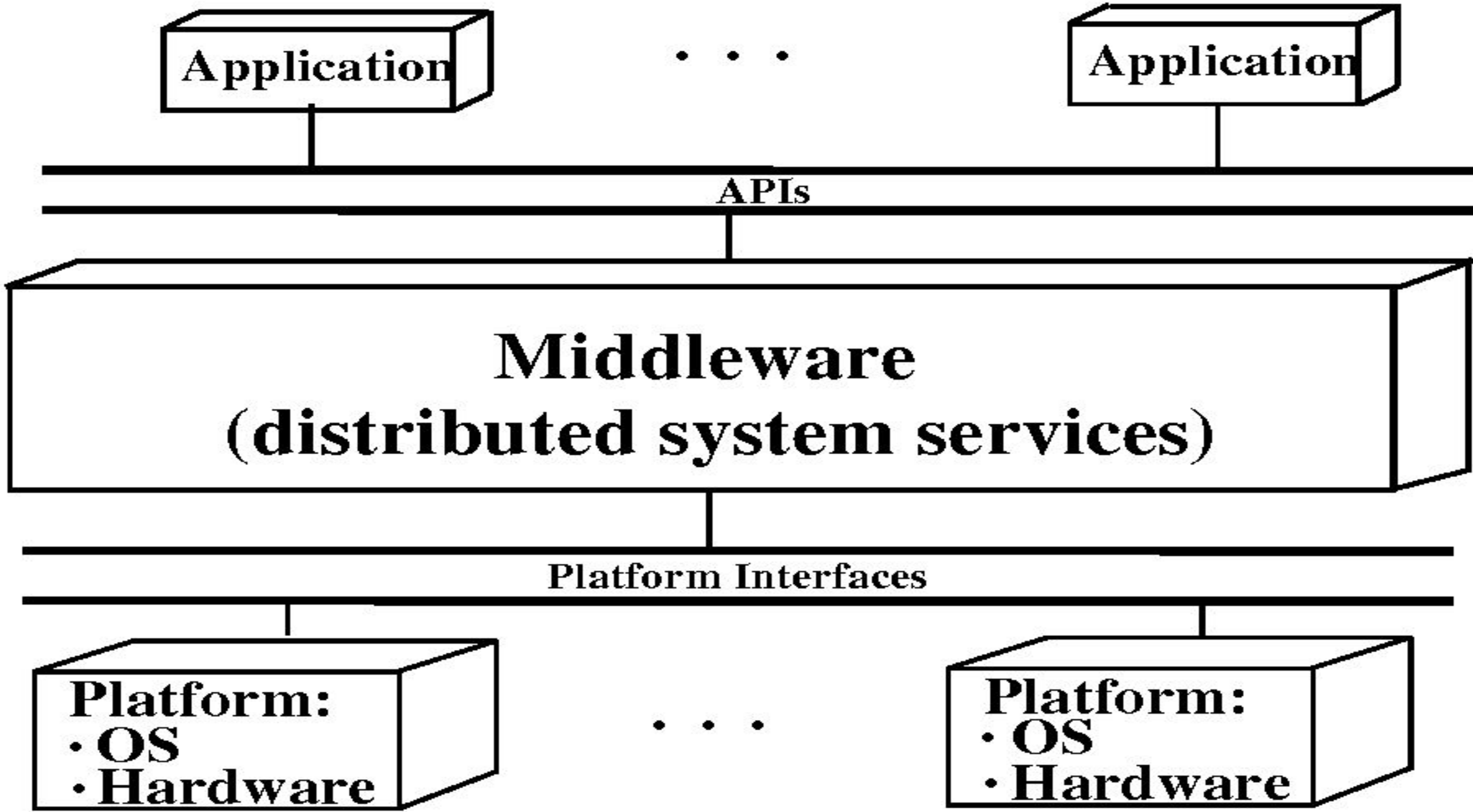
- Most of the data can be **marshaled** from one system to another without losing significance.

4.2. MIDDLEWARE

In distributed systems, middleware is a software component that provides services between two or more applications and can be used by them.

Middleware can be thought of as an application that sits between two separate applications and provides service to both.

M RS6000 and AIX, and Alpha AXP and OpenVMS.



Working of Middleware

Application Communication:

- An application sends a request to another application or service.
- Middleware intercepts this request, processes it (e.g., applies security checks, converts data formats), and forwards it to the intended recipient.
- The recipient processes the request and sends a response back.
- Middleware intercepts the response, processes it, and forwards it back to the original application.

Resource Access:

- An application requests access to a resource (e.g., a database).
- Middleware handles the connection, manages transactions, and ensures secure access.
- It can also manage a pool of connections to optimize resource usage.

Transaction Management:

- Middleware can manage complex transactions that span multiple systems and databases.
- It ensures all parts of the transaction are completed successfully or rolled back if any part fails, maintaining data consistency.

ADVANTAGES

- Middleware supports the scalability of applications by managing resources efficiently and facilitating load balancing and fault tolerance.
- Middleware enables different applications, potentially running on diverse operating systems and platforms, to communicate and work together seamlessly.
- Middleware allows developers to modify and update applications without affecting the underlying hardware or other applications.

- Middleware often includes built-in security features such as authentication, authorization, encryption, and data validation.

- Common services provided by middleware, such as authentication, logging, and messaging, can be reused across different applications, reducing redundancy and saving development time.

CHALLENGES

- Implementing middleware can add complexity to the system architecture.
- Middleware can introduce performance overhead due to the additional processing required for communication, data translation, and other middleware services.
- Middleware solutions, particularly commercial ones, can be expensive.
- Integrating middleware with existing systems and applications can be challenging.
- Relying heavily on middleware can create a dependency on the middleware vendor or technology.

4.3. OBJECT IN DISTRIBUTED SYSTEM

Distributed objects are **reusable software components** that can be distributed and accessed by the user across the network.

These objects can be assembled into distributed applications.

publish-subscribe channels and multicast groups are examples of live distributed objects

A distributed object is an object that can be accessed remotely.

This means that a distributed object can be used like a regular object, but from anywhere on the network.

Distributed objects might be used :

1. to share information across applications or users.

2. to synchronize activity across several machines.

3. to increase performance associated with a particular task.

Local Objects vs. Distributed Objects

● Local objects are those whose methods can only be invoked by a local process, a process that runs on the same computer on which the object exists.

● A distributed object is one whose methods can be invoked by a remote process, a process running on a computer connected via a network to the computer on which the object exists.

HOW DISTRIBUTED OBJECT COMMUNICATE ?

-RMI

-RPC

CORBA

4.4. THE CORBA APPROACH

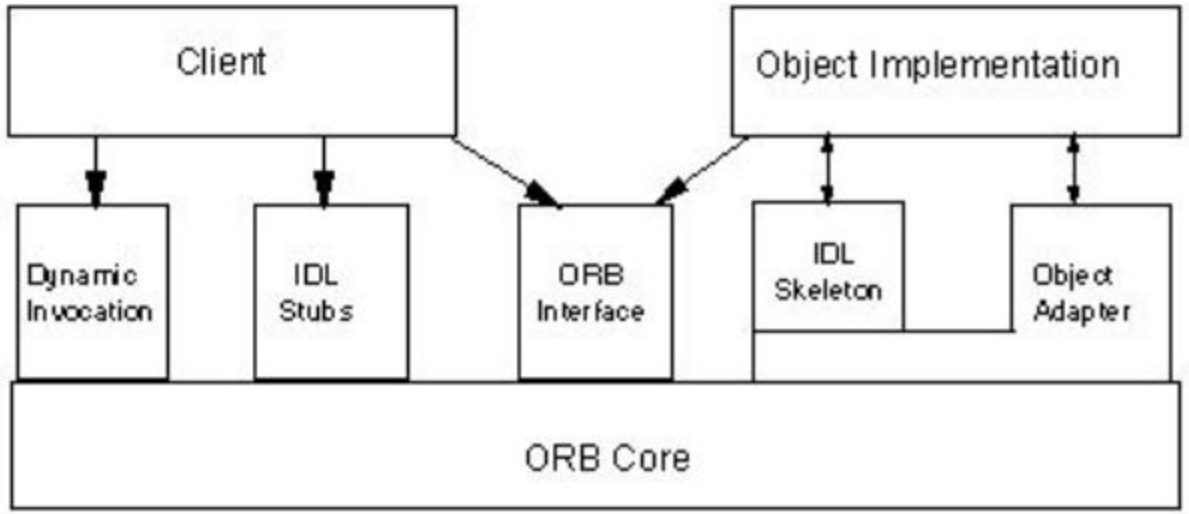
The Common Object Request Broker Architecture (CORBA) describes a messaging mechanism by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects.

CORBA enables collaboration between systems on different operating systems, programming languages, and computing hardware. CORBA uses an object-oriented model although the systems that use the CORBA do not have to be object-oriented.

The essential concept in CORBA is the Object Request Broker (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an object) can request services from a server program or object without having to understand where the server is in a distributed network or what the interface to the server program looks like.

To make requests or return replies between the ORBs, programs use the General Inter-ORB Protocol (GIOP) and, for the Internet, its Internet Inter-ORB Protocol (IIOP). IIOP maps GIOP requests and replies to the Internet's Transmission Control Protocol (TCP) layer in each computer.

CORBA Architecture



The CORBA specification defines an architecture of interfaces and services that must be provided by the ORB, with no implementation details.

These are modular components so different implementations could be used, satisfying the needs of different platforms

The ORB manages the interactions between clients and object implementations. Clients issue requests and invokes methods of object implementations.

There are two basic types of objects in CORBA.

The object that includes some functionality and may be used by other objects is called a service provider.

The object that requires the services of other objects is called the client.

The service provider object and client object communicate with each other independent of the programming language used to design them and independent of the operating system in which they run.

Each service provider defines an interface, which provides a description of the services provided by the client.

Dynamic Invocation - This interface allows for the specification of requests at runtime. This is necessary when the object interface is not known at run-time. Dynamic Invocation works in conjunction with the interface repository.

IDL Stub - This component consists of functions generated by the IDL interface definitions and linked into the program. The functions are a mapping between the client and the ORB implementation. Therefore ORB capabilities can be made available for any client implementation for which there is a language mapping. Functions are called just as if it was a local object.

ORB Interface - The ORB interface may be called by either the client or the object implementation. The interface provides functions of the ORB which may be directly accessed by the client (such as retrieving a reference to an object.) or by the object implementations.

This interface is mapped to the host programming language. The ORB interface must be supported by any ORB. ORB core - Underlying mechanism used as the transport level. It provides basic communication of requests to other sub-components.

IDL Skeleton Interface - The ORB calls method skeletons to invoke the methods that were requested from clients.

Object Adapters (OA) - Provide the means by which object implementations access most ORB services. This includes the generation and interpretation of object references, method invocation, security, and activation.

Requests -The client requests a service from the object implementation. The ORB transports the request, which invokes the method using object adapters and the IDL skeleton

Object Adapters - Object Adapters (OA) are the primary ORB service providers to object implementations. OA has a public interface that is used by the object implementation and a private interface that is used by the IDL skeleton.

INVOCATION IN CORBA

Static and Dynamic Invocation

The CORBA ORB in the BEA Tuxedo product supports two types of client/server invocations: static and dynamic.

In both cases, the CORBA client application performs a request by gaining access to a reference for a CORBA object and invoking the operation that satisfies the request.

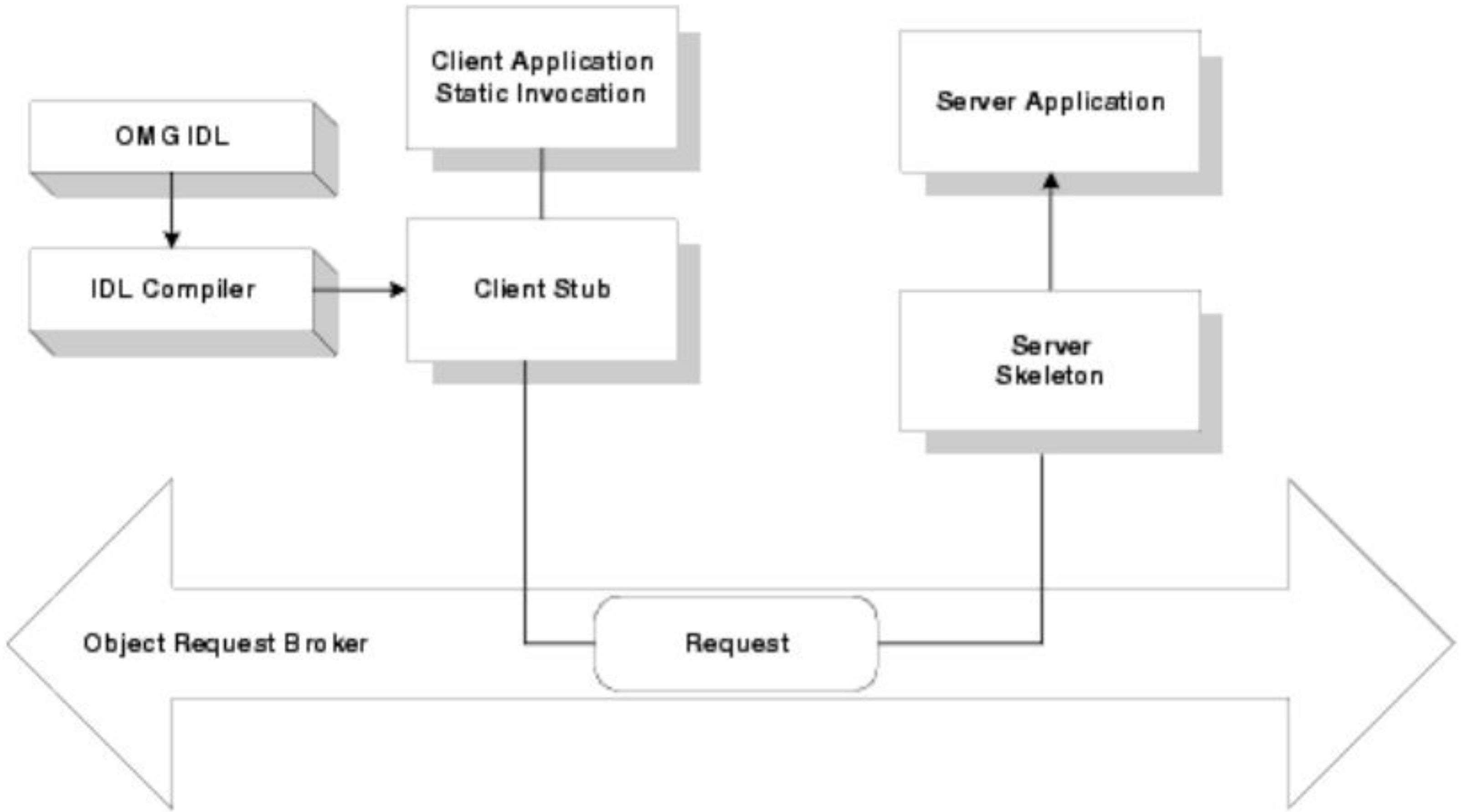
The CORBA server application cannot tell the difference between static and dynamic invocations.

When using static invocation, the CORBA client application invokes operations directly on the client stubs.

Static invocation is the easiest, most common type of invocation.

The stubs are generated by the IDL compiler.

Static invocation is recommended for applications that know at compile time the particulars of the operations they need to invoke and can process within the synchronous nature of the invocation. The figure illustrates static invocation.


OMG IDL
IDL Compiler
Client Application
Static Invocation
Client Stub
Server Application
Server
Skeleton
Object Request Broker
Request

While dynamic invocation is more complicated, it enables your CORBA client application to invoke operations on any CORBA object without having to know the CORBA object's interfaces at compile time.

The figure illustrates the dynamic invocation.

When using dynamic invocation, the CORBA client application can dynamically build operation requests for a CORBA object interface that has been stored in the Interface Repository.

CORBA server applications do not require any special design to be able to receive and handle dynamic invocation requests.

Dynamic invocation is generally used when the CORBA client application requires deferred synchronous communication, or by dynamic client applications when the nature of the interaction is undefined.

4.5. CORBA SERVICES

Naming and Trading Services:

● The basic way an object reference is generated is at the creation of the object when the reference is returned.

● Object references can be stored together with associated information (e.g. names and properties).

● The naming service allows clients to find objects based on names.

● The trading service allows clients to find object based on their properties.

☞ Transaction Management Service: provides two-phase commit coordination among recoverable components using transactions.

☞ Concurrency Control Service: provides a lock manager that can obtain and free locks for transactions or threads.

☞ Security Service: protects components from unauthorized users; it provides authentication, access control lists, confidentiality, etc.

☞ Time Service: provides interfaces for synchronizing time; provides operations for defining and managing time-triggered events.