CORBA APPROACH

4.4. THE CORBA APPROACH

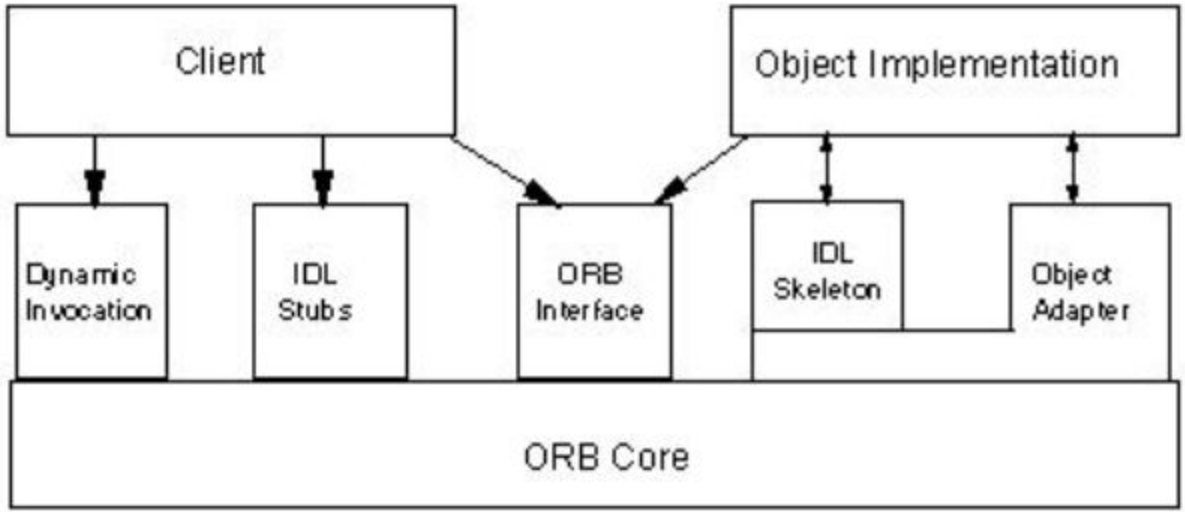
The Common Object Request Broker Architecture (CORBA) describes a messaging mechanism by which objects distributed over a network can communicate with each other irrespective of the platform and language used to develop those objects.

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.

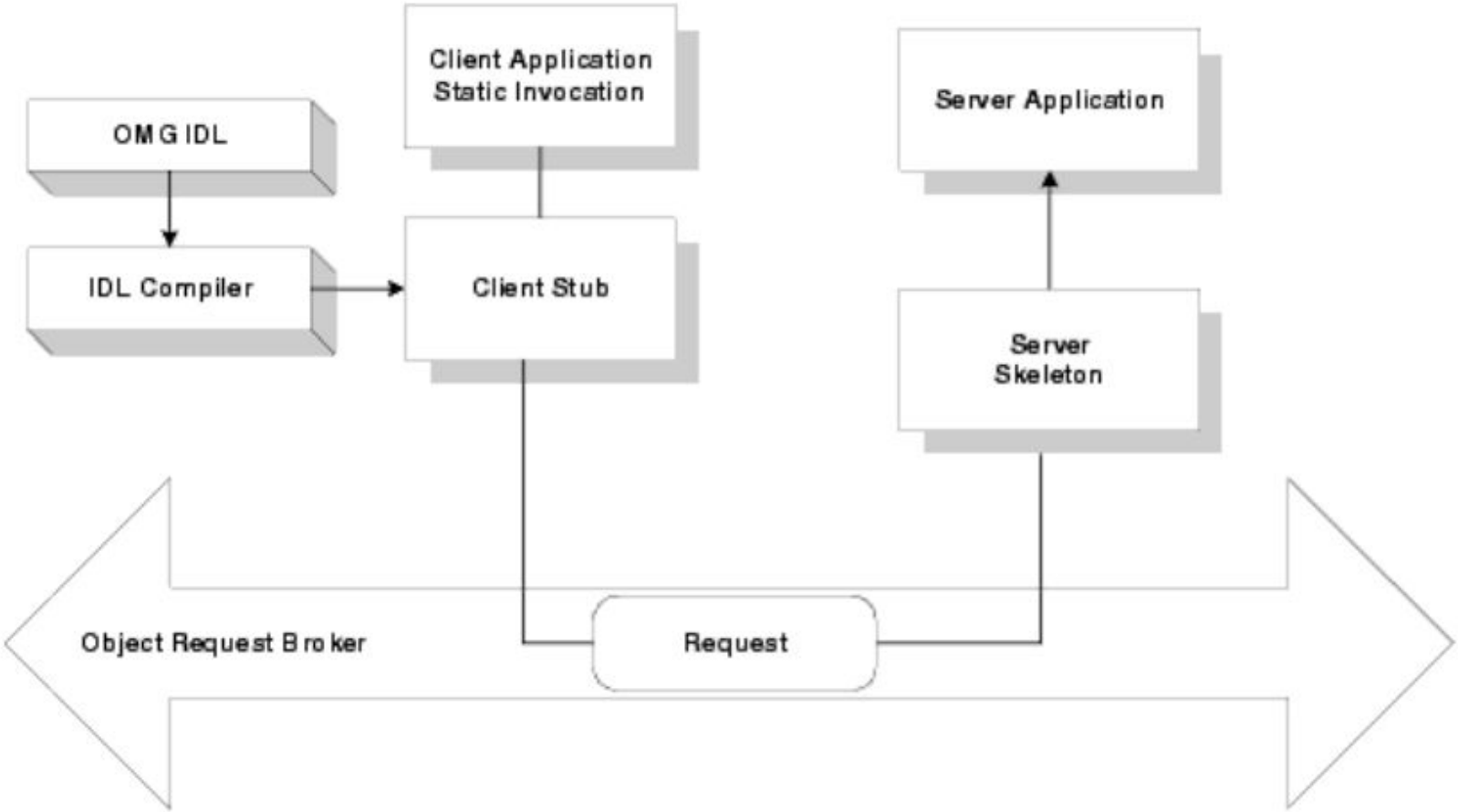
The CORBA server application cannot tell the difference between static and dynamic invocations.

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.


Client Application
Static Invocation
Server Application
OMG IDL
IDL Compiler
Client Stub
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.