CMR
INSTITUTE OF
TECHNOLOGY

USN | | | | | | | | | | |

Internal Assessment Test - I

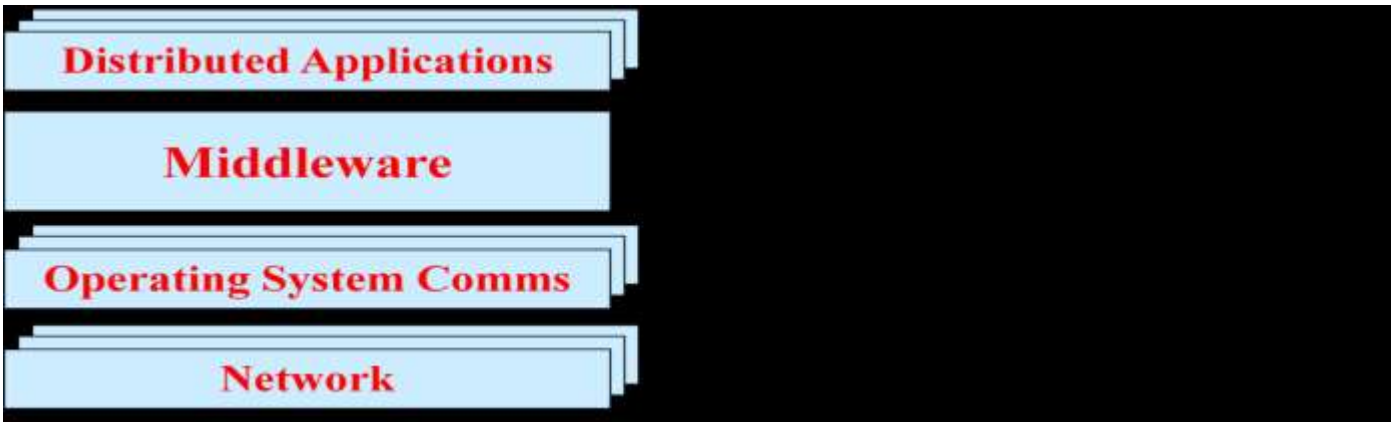
Sub:	WEB SERVICES solutions							Code:	16SCS254
Date:	27 / 03 / 2017	Duration:	90 mins	Max Marks:	50	Sem:	II	Branch:	CSE-MTECH

1. Define middleware? Explain the different types of middleware? (5)

Middleware : Middleware is connectivity software that provides a mechanism for processes to interact with other processes running on multiple networked machines.

Middleware Application Programming Interfaces provide a more functional set of capabilities than the OS and network services provide on their own

- Hides complexity and heterogeneity of distributed system
- Bridges gap between low-level OS communications and programming language abstractions
- Layer between OS and distributed applications



Type of Middleware:

- There are four basic types of middleware
- Transaction Processing Monitor (TP)

Object monitors

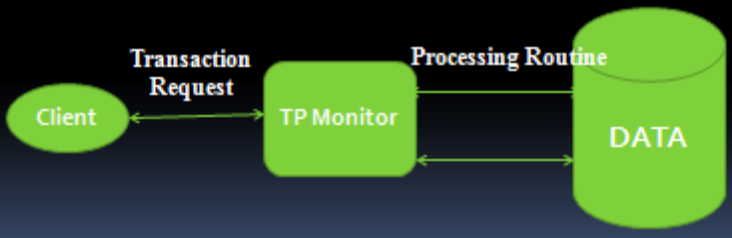
- Remote Procedure Call (RPC)
- Message-Oriented Middleware (MOM)

Message broker

- Object Request Broker (ORB)

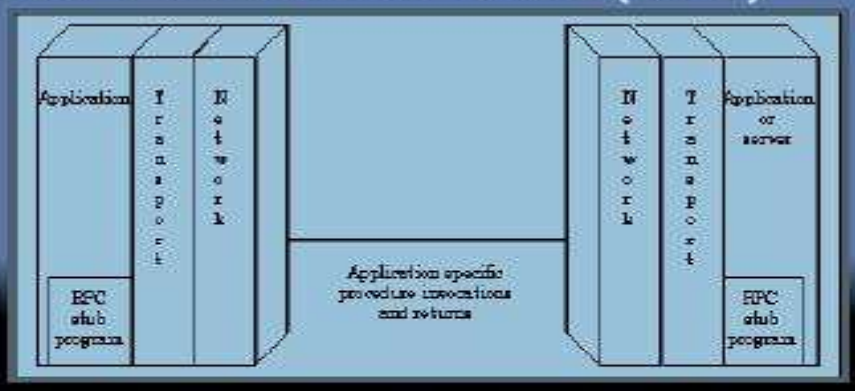
Transaction Processing Monitor (TP)

- TP can provide the following
- – control transaction applications
- – provide business logic/rules
- – database updates



Transaction Processing Architecture

Remote Procedure call(RPC)



- RPC is a client/server mechanism that allows the program to be distributed across multiple platforms.
- RPC's reduce the complexity of a system that span multiple OS and network protocols by OS and network interface details from the programmer

b) Write a short note on Message-Oriented Middleware (5)

- Asynchronous communication is in the heart of Message-Oriented Middleware (MOM)
  - RPC also had an extension to cope with asynchronous calls
  - TP Monitors introduced queues to implement message-based interactions.
- Modern MOM is descendent of queuing systems found in TP Monitors
  - Used for batch processing but later switched to cope with the interoperability issues in the context of multiple heterogeneous systems and programming languages.
- Major approach to integrate information systems and applications today is by relying on MOM
- Current solutions in the area include IBM Web Sphere MQ, Microsoft Message Queuing (MSQM), … but also CORBA has it's service.

- Asynchronous communication is in the heart of Message-Oriented Middleware (MOM)
  - RPC also had an extension to cope with asynchronous calls
  - TP Monitors introduced queues to implement message-based interactions.
- Modern MOM is descendent of queuing systems found in TP Monitors
  - Used for batch processing but later switched to cope with the interoperability issues in the context of multiple heterogeneous systems and programming languages.
- Major approach to integrate information systems and applications today is by relying on MOM
- Current solutions in the area include IBM Web Sphere MQ, Microsoft Message Queuing (MSQM), … but also CORBA has it's service.

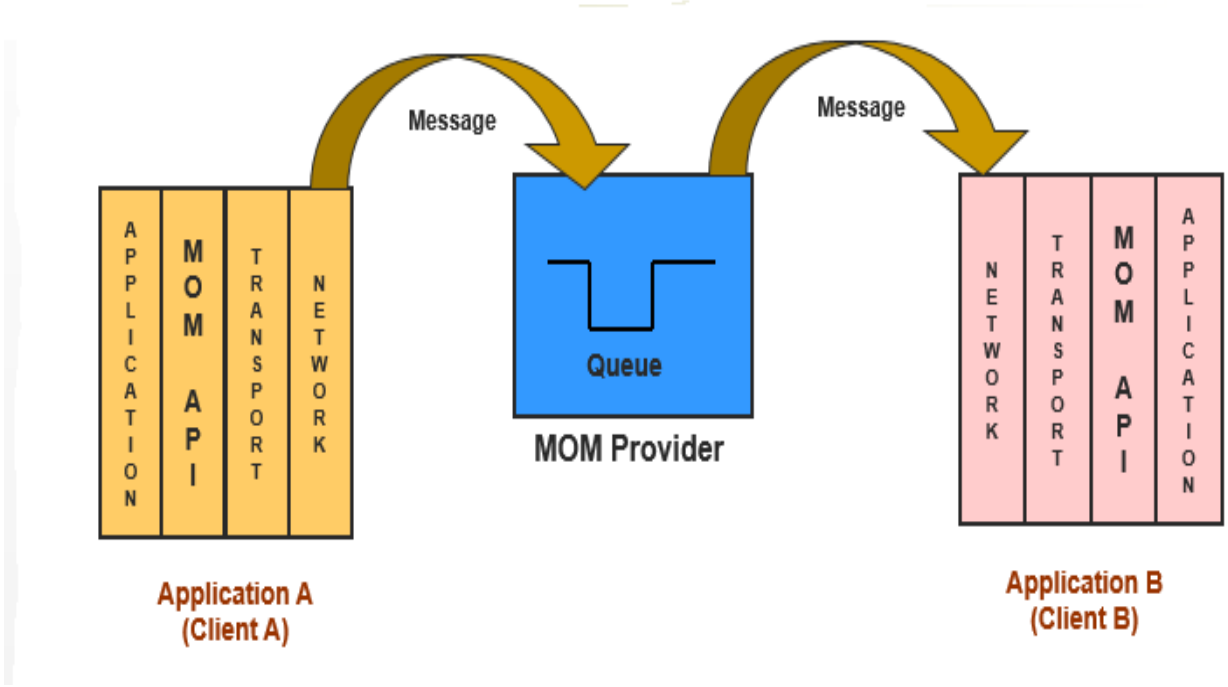
- Asynchronous communication is in the heart of Message-Oriented Middleware (MOM)
  - RPC also had an extension to cope with asynchronous calls
  - TP Monitors introduced queues to implement message-based interactions.
- Modern MOM is descendent of queuing systems found in TP Monitors
  - Used for batch processing but later switched to cope with the interoperability issues in the context of multiple heterogeneous systems and programming languages.
- Major approach to integrate information systems and applications today is by relying on MOM
- Current solutions in the area include IBM Web Sphere MQ, Microsoft Message Queuing (MSQM), ... but also CORBA has it's service.

Asynchronous interaction
- •Client and server are only loosely coupled
- •Messages are queued
- •Good for application integration

Support for reliable delivery service
- •Keep queues in persistent storage

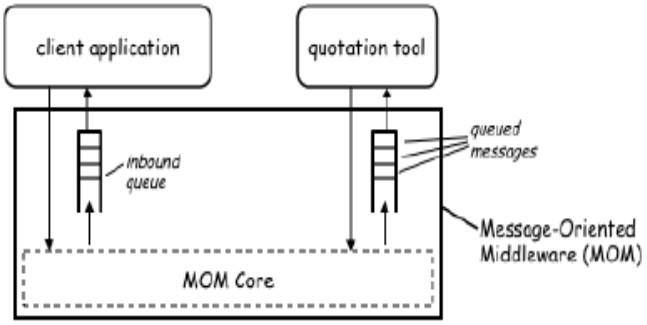
Processing of messages by intermediate message server(s)
- •May do filtering, transforming, logging, …
- •Networks of message servers

Natural for database integration



- It refers to an interaction paradigm where clients and service providers communicate by exchanging messages.
- Message is a structured data set characterized by its type and a set of pairs consisting of names and values.
- Client and service provider must agree on the set of message types exchanged during the communication.
- MOM supports message-based interoperability.
- Difference between client and server is here blurred.

- Messages sent by a MOM client are placed into a queue.
- Queue is identified by a name and possibly bound to a specific intended recipient.
- Recipient picks up and processes the message when suitable.
- More robust on failures, flexible in terms of performance optimizations:
  - Queues may be shared between applications.
  - Messages may have priorities.



2. Describe how the Binding happen in RPC along with its Working(10)

Remote Procedure Call (RPC) is a high-level model for client-server communication.

- RPC enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high-level languages.
- Examples: File service, Authentication service.

RPC Model

- Fundamental idea: –
  - Server process exports an *interface* of procedures or functions that can be called by client programs similar to library API, class definitions, etc.
- Clients make local procedure/function calls
  - *As if* directly linked with the server process
  - Under the covers, procedure/function call is converted into a message exchange with remote server process

Ordinary procedure/function call

count = read(fd, buf, nbytes)

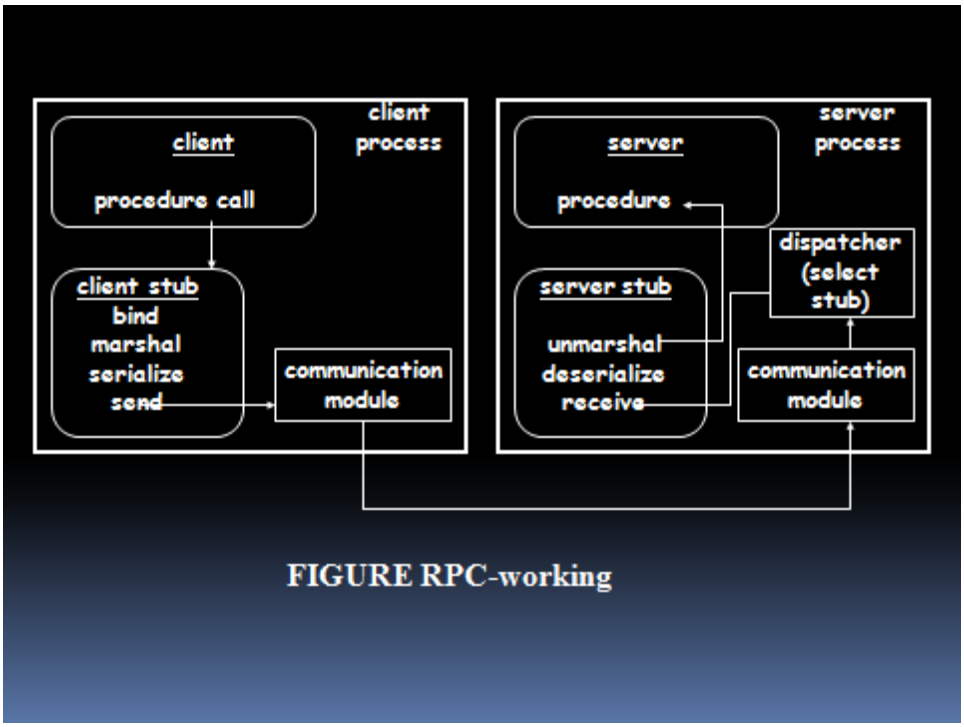
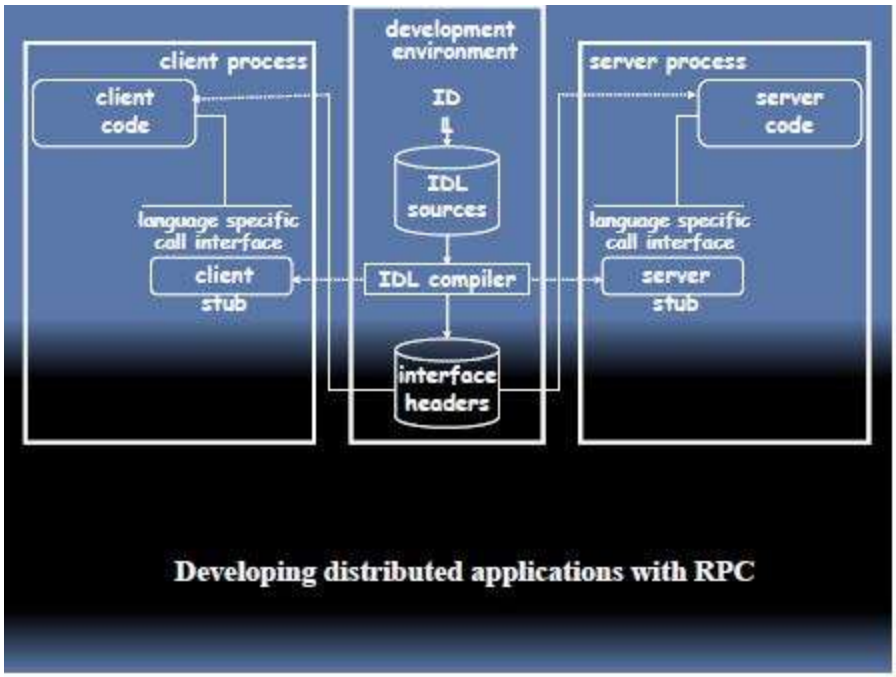


FIGURE RPC-working

Solution — a pair of *Stubs*

- **Client-side stub**
  - Looks like local server function
  - Same interface as local function
  - Bundles arguments into message, sends to server-side stub
  - Waits for reply, un-bundles results
  - returns
- **Server-side stub**
  - Looks like local client function to server
  - Listens on a socket for message from client stub
  - Un-bundles arguments to local variables
  - Makes a local function call to server
  - Bundles result into reply message to client stub



RPC Model

A server defines the service interface using an *interface definition language* (IDL)

- the IDL specifies the names, parameters, and types for all client-callable server procedures
  - A *stub compiler* reads the IDL declarations and produces two *stub functions* for each server function
  - *Server-side* and *client-side*
- **Linking:–**
  - Server programmer implements the service's functions and links with the *server-side* stubs
  - Client programmer implements the client program and links it with *client-side* stubs
- **Operation:–**
  - Stubs manage all of the details of remote communication between client and server
- **A *client-side stub* is a function that looks to the client as if it were a callable server function**
  - I.e., same API as the server's implementation of the function

- A *server-side stub* looks like a caller to the server
  - I.e., like a hunk of code invoking the server function
- The client program thinks it's invoking the server
  - but it's calling into the client-side stub
- The server program thinks it's called by the client
  - but it's really called by the server-side stub
- The stubs send messages to each other to make the RPC happen transparently (almost!)
- A *client-side stub* is a function that looks to the client as if it were a callable server function
  - I.e., same API as the server's implementation of the function
- A *server-side stub* looks like a caller to the server
  - I.e., like a hunk of code invoking the server function
- The client program thinks it's invoking the server
  - but it's calling into the client-side stub
- The server program thinks it's called by the client
  - but it's really called by the server-side stub
- The stubs send messages to each other to make the RPC happen transparently (almost!)

Marshalling Arguments

- *Marshalling* is the packing of function parameters into a message packet
  - The RPC stubs call type-specific functions to marshal or unmarshal the parameters of an RPC
    Client stub marshals the arguments into a message
    Server stub unmarshals the arguments and uses them to invoke the service function
  - on return:
    the server stub marshals return values
    the client stub unmarshals return values, and returns to the client program

RPC Binding

RPC BINDING

Binding is the process of connecting the client to the server

- the server, when it starts up, exports its interface

identifies itself to a *network name server*

tells *RPC runtime* that it is alive and ready to accept calls

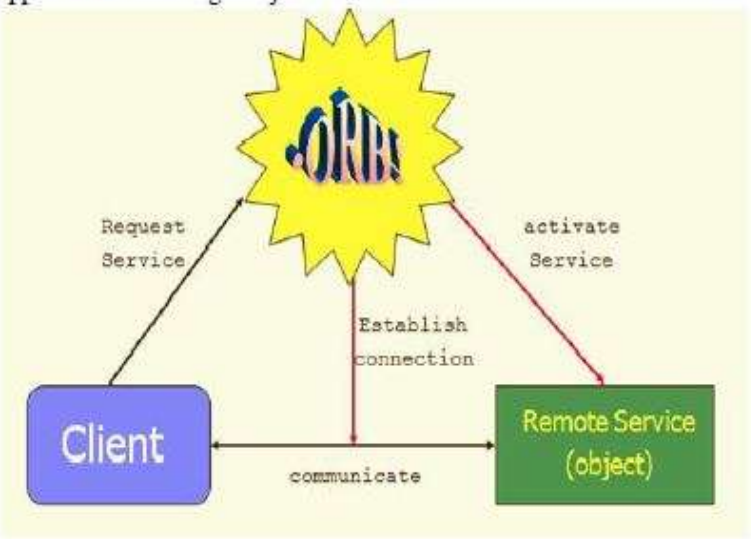
- the client, before issuing any calls, imports the server

RPC runtime uses the name server to find the location of the server and establish a connection

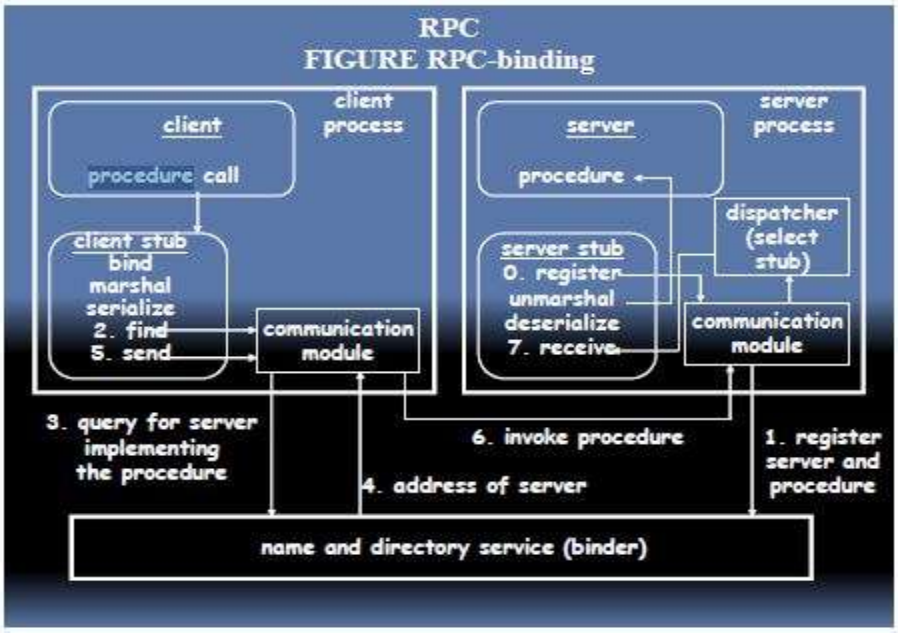
- The import and export operations are explicit in the server and client programs

3. Describe Briefly about CORBA and ORB?(10)

Ans: Common Object Request Broker: An Object Broker is a middleware entity that matches up client applications with target objects.



A target object is a software entity that provides some service to a client software application. It may be located on the same machine as the client, or half a world away. The client doesn't need to know. It simply tells the Object Broker (also known as Object Request Broker, or ORB)



Introduction to CORBA:

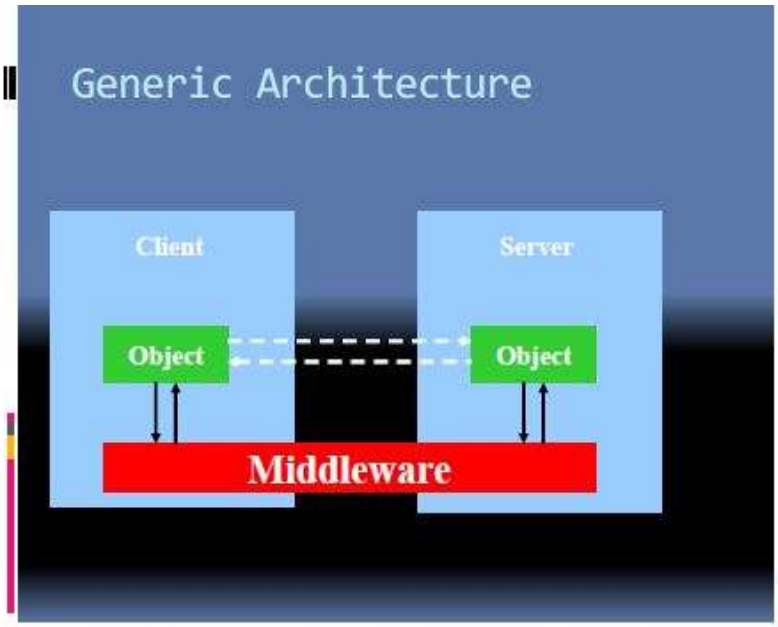
- The Object Management Group (OMG) was formed in 1989. Its aims were:
  - to make better use of distributed systems
  - to use object-oriented programming
  - to allow objects in different programming languages to communicate with one another
- The object request broker (ORB) enables clients to invoke methods in a remote object
- CORBA is a specification of an architecture supporting this.
  - CORBA 1 in 1990 and CORBA 2 in 1996.

(PORTABLE) OBJECT ADAPTER (POA)

- Register class implementations
- Creates and destroys objects
- Handles method invokation
- Handles client authentication and access control

OBJECT REQUEST BROKER (ORB)

- Communication infrastructure sending messages between objects
- Communication type:
  - GIOP (General Inter-ORB Protocol)
  - IIOP (Internet Inter-ORB Protocol) (GIOP on TCP/IP)



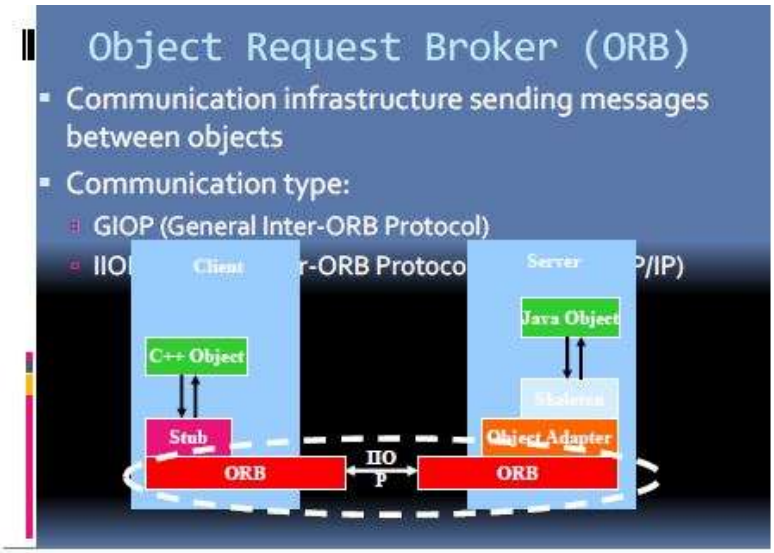
- Remote-object: object implementation resides in server's address space

STUB

- Provides interface between client object and ORB
- Marshalling: client invocation
- Unmarshalling: server response

SKELETON

- Provides iterface between server object and ORB
- Unmarshaling: client invocation
- Marshaling: server response

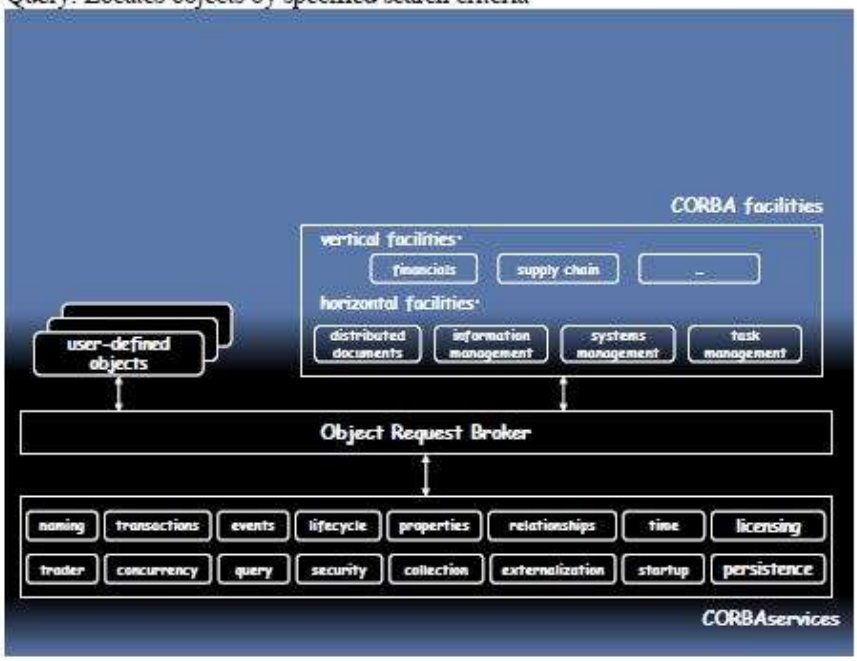


INTERFACE DEFINITION LANGUAGE (IDL)

- Describes interface
- Language independent
- Client and server platform independent

EXAMPLE OF CORBA SERVICES

- Naming: Keeps track of association between object names and their reference. Allows ORB to locate referenced objects
- Life Cycle: Handles the creation, copying, moving, and deletion objects
- Trader: A "yellow pages" for objects. Lets you find them by the services they provide
- Event: Facilitates asynchronous communications through events
- Concurrency: Manages locks so objects can share resources
- Query: Locates objects by specified search criteria



4. Explain the External Architecture of a web service argument with peer-to peer protocol execution(10)

- Case for the external middleware is not clear
  - Who owns the middleware?
  - Where to locate it?
  - How to trust to the provided middleware services?

- Two solutions to solve the problem:
  1. Implement middleware as P2P system
     - All participants cooperate to provide the services
     - Reliability and trustworthiness is questionable (e.g., name and directory services)
  2. Introduce intermediaries or brokers acting as necessary middleware
     - Part of the middleware can reside at different locations

- Currently only name and directory services (UDDI) is standardized and "used" in practice.

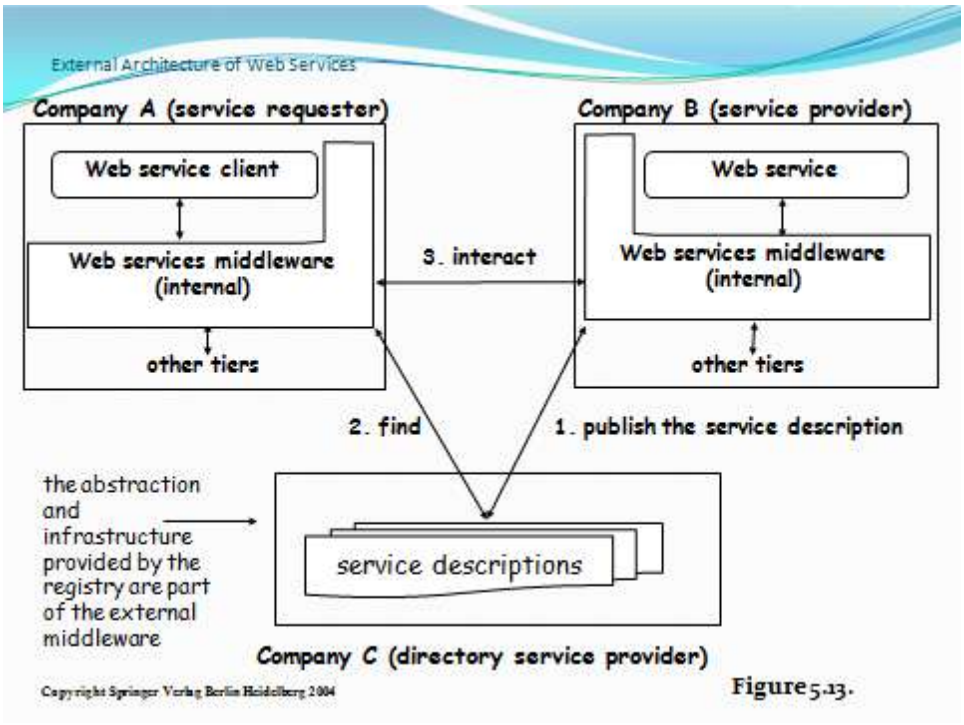
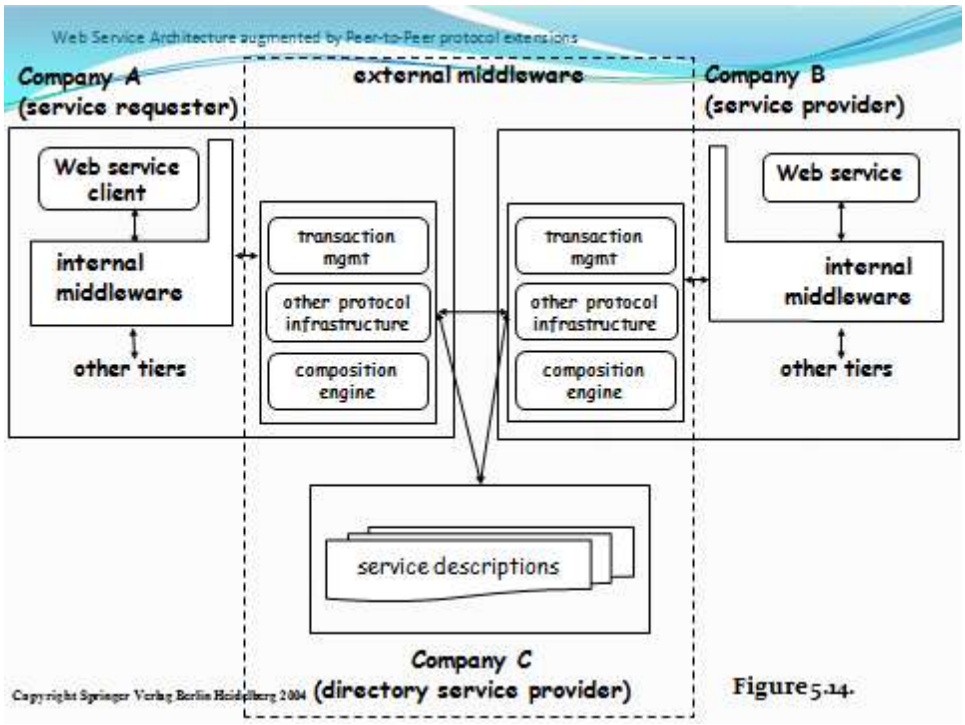


Figure 5.13.

- But where are the other middleware features (transaction management supported by TP monitors, services provided by CORBA)?

- Centralized transaction broker is theoretically possible and technically feasible but runs into various problems
  - Standard way of running transactions accepted by everyone so that transactional semantics is not violated.
  - All participants trust the broker (highly improbable).

- Alternative is to implement the transaction broker as a P2P system
  - Each service requester has its own transaction manager.
  - Functionality provided by this solution is a subset of the functionality offered by conventional middleware systems.



5. Explain about TP Monitors and Functionality of a TP Monitor (10)

Ans: The solution to this limitation is to make RPC calls transactional, that is, instead of providing plain RPC, the system should provide TRPC

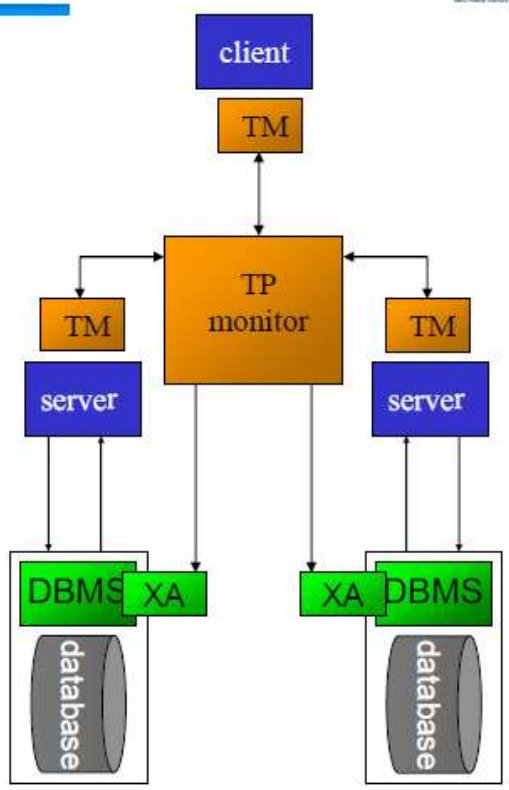
- What is TRPC?
  - same concept as RPC plus …
  - additional language constructs and run time support (additional services) to bundle several RPC calls into an atomic unit
  - usually, it also includes an interface to databases for making end-to-end transactions using the XA standard (implementing 2 Phase Commit)

- and anything else the vendor may find useful (transactional callbacks, high level locking, etc.)

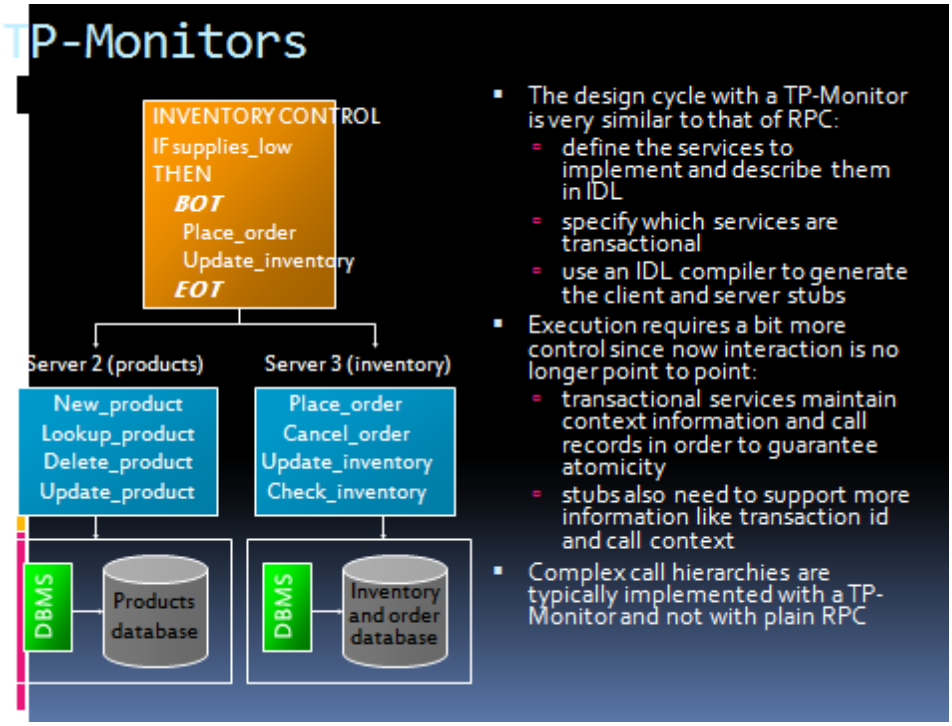
Simplifying things quite a bit, one can say that, historically, TP-Monitors are RPC based systems with transactional support. We have already seen an example of this:

Transactional RPC

- The limitations of RPC can be resolved by making RPC calls transactional. In practice, this means that they are controlled by a 2PC protocol
- As before, an intermediate entity is needed to run 2PC (the client and server could do this themselves but it is neither practical nor generic enough)
- This intermediate entity is usually called a transaction manager (TM) and acts as intermediary in all interactions between clients, servers, and resource managers
- When all the services needed to support RPC, transactional RPC, and additional features are added to the intermediate layer, the result is a TP-Monitor



- Concept of transaction is developed in the context of Database Management Systems (DBMS).
- Transaction represents a set of operations characterized by the so called ACID properties.
- TRPC provides a possibility to enforce ACID properties when dealing with data distributed across multiple (and heterogeneous) systems.
- Procedure calls enclosed within the transactional brackets are an atomic unit of work.
  - Beginning of Transaction (BOT) and End of Transaction (EOT) – RPC call.
  - Infrastructure guarantees their atomicity.
- Transaction Management module is responsible to coordinate interactions between clients and servers.



6.Explain the Two Facets of Web Services Architecture ?(10)

Ans: web services: A web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer. This interoperability (e.g., between Java and Python, or Windows and Linux applications) is due to the use of open standards**The Two Faces of Web Service Architectures:**

- Web services as a way to expose internal operations of a company
  - System receives requests through the Web and passes them to the underlying IT system.
  - The problems are analogous to those encountered in conventional middleware.
  - This is *internal middleware* for Web services (term *internal architecture* is used to refer to organization and structure of the internal middleware).

- Web services as a way to integrate systems across the Internet.
  - Middleware infrastructure is needed to integrate different Web services.
  - This is *external middleware* for Web services (term *external architecture* is used to refer to organization and structure of the external middleware).
  - External architecture has three components
    - Centralized brokers – message routing and providing support for interactions (logging, transactional guaranties, name and directory services, etc).
    - Protocol infrastructure – coordinating interactions between Web services in distributed settings.
    - Service composition infrastructure – definition and execution of composite services.

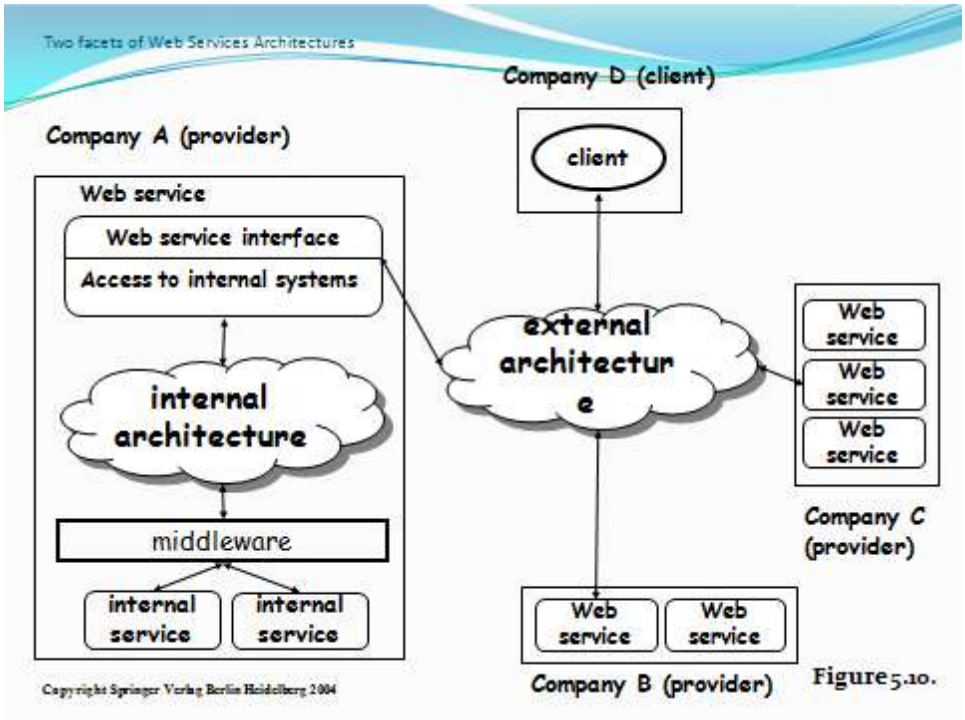


Figure 5.10.

External Architecture

- Case for the external middleware is not clear
  - Who owns the middleware?
  - Where to locate it?
  - How to trust to the provided middleware services?

- Two solutions to solve the problem:
  1. Implement middleware as P2P system
     - All participants cooperate to provide the services
     - Reliability and trustworthiness is questionable (e.g., name and directory services)
  2. Introduce intermediaries or brokers acting as necessary middleware
     - Part of the middleware can reside at different locations

- Currently only name and directory services (UDDI) is standardized and "used" in practice.

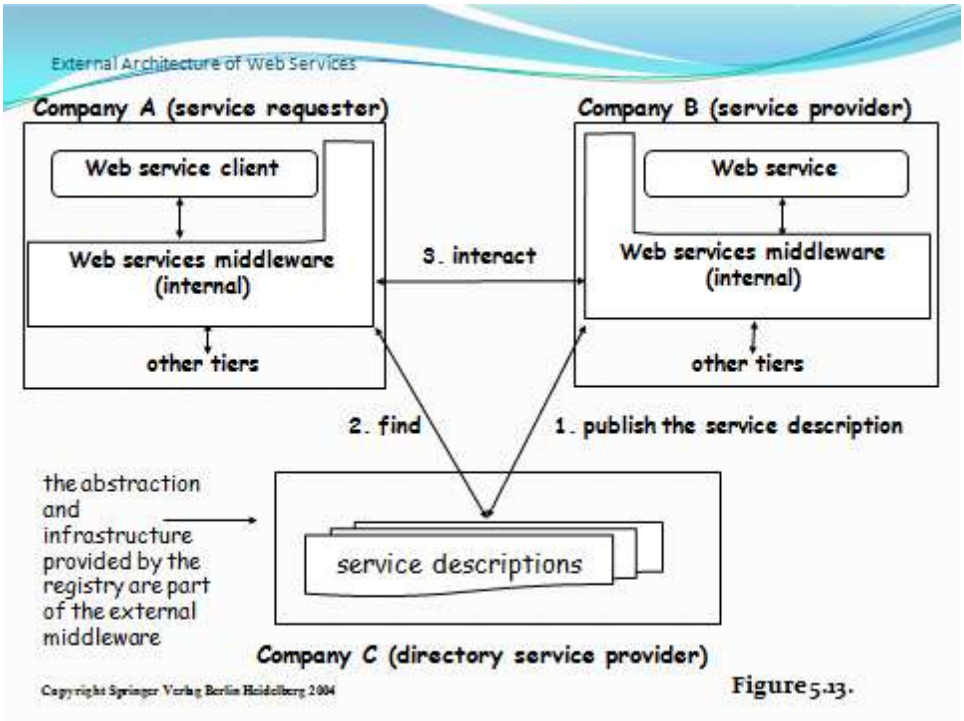


Figure 5.13.

- But where are the other middleware features (transaction management supported by TP monitors, services provided by CORBA)?

- Centralized transaction broker is theoretically possible and technically feasible but runs into various problems
  - Standard way of running transactions accepted by everyone so that transactional semantics is not violated.
  - All participants trust the broker (highly improbable).

- Alternative is to implement the transaction broker as a P2P system
  - Each service requester has its own transaction manager.
  - Functionality provided by this solution is a subset of the functionality offered by conventional middleware systems.

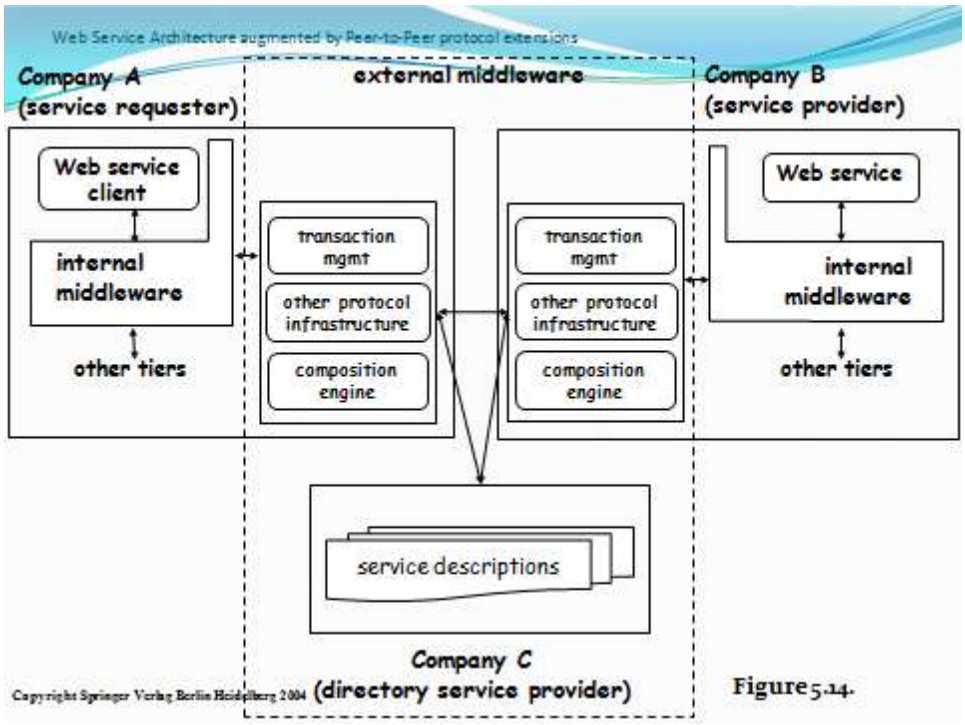


Figure 5.14.

7.a) Explain CORBA Encapsulation with a Dynamic Service Selection and Invocation (5)

CORBA allows client applications to dynamically discover new objects, retrieve their interfaces, and construct invocations of this object on the fly, even if no stub has been previously

generated and linked to the client. This capability is based on two components: the *interface repository* and the *dynamic invocation interface.* The interface repository stores IDL definitions for all the objects known to the ORB. Applications can access the repository to browse, edit, or delete IDL interfaces. The dynamic invocation interface provides operations such as *getinterface* and *createrequest* that can be used by clients to browse the repository and dynamically construct the method invocation based on the newly discovered interface.

Constructing dynamic invocations is in fact very difficult. One problem is that, to search for services, the client object must understand the meaning of the service properties, which in turn requires a shared ontology among clients and service providers. Furthermore, if the client has not been specifically implemented to interact with a certain service, it is difficult that it is able to figure out what the operations of the newly discovered service do, what is the exact meaning of their parameters, and in what order they should be invoked to obtain the desired functionality.

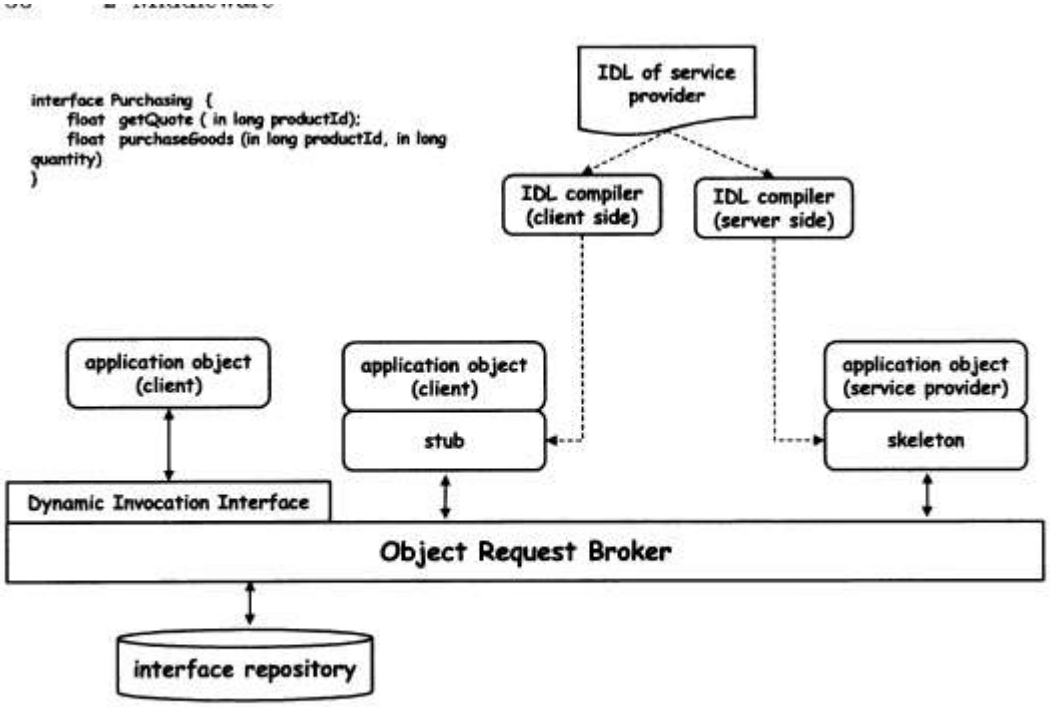


Fig. 2.10. IDL specifications are compiled into skeletons on the server side and into stubs on the client side

B. Explain about 2Phase Commit protocol(2PC). (5)

2PC is the standard mechanism for guaranteeing atomicity in distributed information systems. In 2PC, the transaction manager executes the commit in two phases: in the first phase, it contacts each server involved in the transaction by sending a *prepare to commit* message, asking whether the server is ready to execute a commit. If the server successfully completed the procedure invoked by the TRPC, it answers that it is *ready to commit.* By doing this, the server

guarantees that it will be able to commit the procedure even if failures occur. If the server could not complete the TRPC, it replies *abort.* In the second phase, the transaction manager examines all the replies obtained and, if all of them are *ready to commit,* it then instructs each server to commit the changes performed as part of the invoked procedure. If at least one resource manager replied *abort* (or failed to reply within a specified time limit), then the transaction manager requests all servers to abort the transaction.

Fault tolerance in 2PC is achieved through *logging* (writing the state of the protocol to persistent storage). By consulting such log entries, it is possible to reconstruct the situation before a failure occurred and recover the system. What is logged and when depends on the flavor of 2PC used *(presumed nothing, presumed abort, presumed commit,*) but it is an important factor in terms of performance, since it must be done for each transaction executed and since logging implies writing to the disk (which is a time-consuming operation). 2PC may block a transaction if the coordinator fails after sending the *prepare to commit* messages but before sending a commit message to all participants.