DISTRIBUTED HETEROGENEOUS APPLICATIONS AND CORBA

1. Heterogeneity in Distributed Systems

2. Middleware

3. Objects in Distributed Systems

4. The CORBA Approach

5. Components of a CORBA Environment

6. CORBA Services

Heterogeneity in Distributed Systems

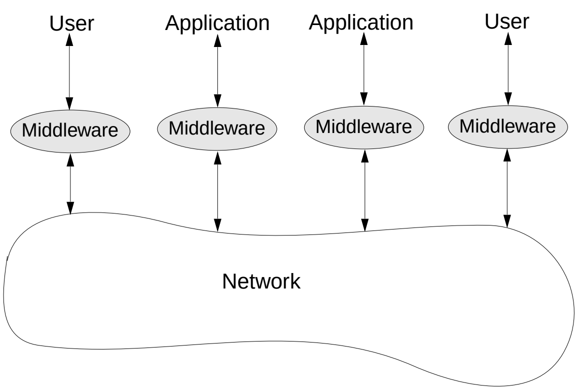
☞ Distributed applications are typically heterogeneous:
- different hardware: mainframes, workstations, PCs, servers, etc.;
- different software: UNIX, MS Windows, IBM OS/2, Real-time OSs, etc.;
- unconventional devices: teller machines, telephone switches, robots, manufacturing systems, etc.;
- diverse networks and protocols: Ethernet, FDDI, ATM, TCP/IP, Novell Netware, etc.

☞ Why?
- Different hardware/software solutions are considered to be optimal for different parts of the system.
- Different users which have to interact are deciding for different hardware/software solutions/vendors.
- **Legacy systems**.

Middleware

☞ A key component of a heterogeneous distributed client-server environment is *middleware*.

- *Middleware* is a set of services that enable applications and end users to interact with each other across a heterogeneous distributed system. Middleware software resides above the network and below the application software.

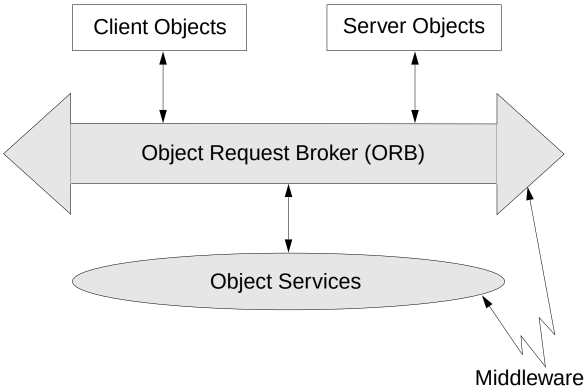


Middleware (cont'd)

- Middleware should make the network transparent to the applications and end users ⇒ users and applications should be able to perform the same operations across the network that they can perform locally.
- Middleware should hide the details of computing hardware, OS, software components across networks.
- Different kind of software qualifies, to certain extent, as middleware:
  - File-transfer packages (FTP) and email;
  - Web browsers;
  - CORBA

Objects in Distributed Systems

- A distributed application can be viewed as a collection of objects (user interfaces, databases, application modules, customers).



Objects in Distributed Systems (cont'd)

- Objects are data surrounded by code; each one has its own attributes and methods which define the behavior of the object; objects can be clients, servers, or both.

- Middleware:

  - Object brokers allow objects to find each other in a distributed system and interact with each other over a network; they are the backbone of the distributed object-oriented system.

  - Object services allow to create, name, move, copy, store, delete, restore, and manage objects.

- Modeling in terms of OO concepts does not necessarily imply use of OO programming languages for implementation or the use of OO database managers as part of the system.

Objects in Distributed Systems (cont'd)

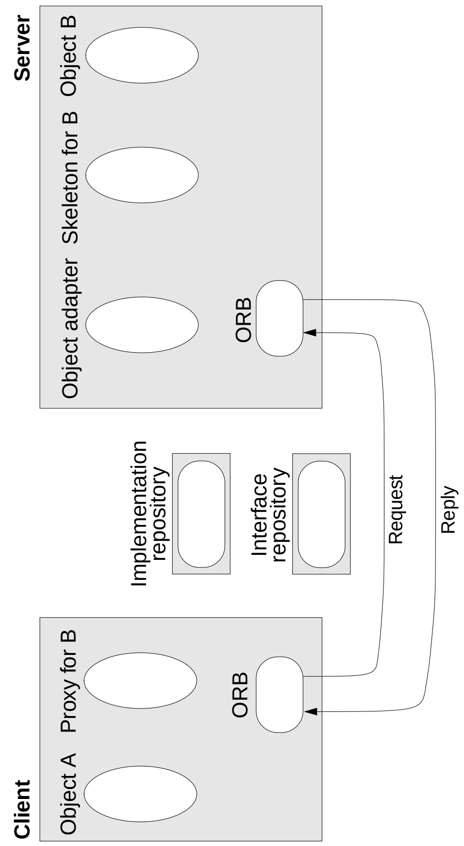
If we relate to the picture in Lecture 2/3, slide 32, which explains RMI, we can recognize some components:

- We have the client and server objects.
- We have the skeleton and proxy, which are on the border between middleware and application.
- The *communication module* and the *remote reference module* are part of the ORB.

☞ Additional components which are part of the middleware:

- Object adapter
- Implementation repository
- Interface repository.

Objects in Distributed Systems (cont'd)



Interface Definition Language

☞ An **interface** specifies the API (Application Programming Interface) that the clients can use to invoke operations on objects:
- the set of operations
- the parameters needed to perform the operations.

- One or more interfaces can be defined for an object. Such, different interfaces can be defined for different classes of users of the same object.
- Interfaces are defined by using an **interface definition language** (IDL).
  CORBA IDL is an example of such a language.

Interface Definition Language (cont'd)

☞ Middleware products (such as CORBA) provide interface compilers that parse the IDL description of the interface. Such a compiler produces the code which represents:
- the classes corresponding to the *proxies* (in the language of the client).
- the classes corresponding to the *skeletons* (in the language of the server).

- If the client or the server are not in an object oriented language, the compiler generates a client stub (instead of proxy class) respectively server stub (instead of skeleton class).

☞ IDLs are declarative languages; they do not specify any executable code, but only declarations.

☞ IDLs should be implementation language independent ⇒ the interface is defined independent of the language in which the server and its clients are implemented.

Language mappings have to be defined which allow to compile the IDL interface and to generate proxies and skeletons in the implementation languages of the clients and of the server respectively.

CORBA

☞ Object Management Group (OMG): a non-profit industry consortium formed in 1989 with the goal to develop, adopt, and promote standards for the development of distributed heterogeneous applications.

☞ One of the main achievements of OMG is the specification of a Common Object Request Broker Architecture (CORBA).

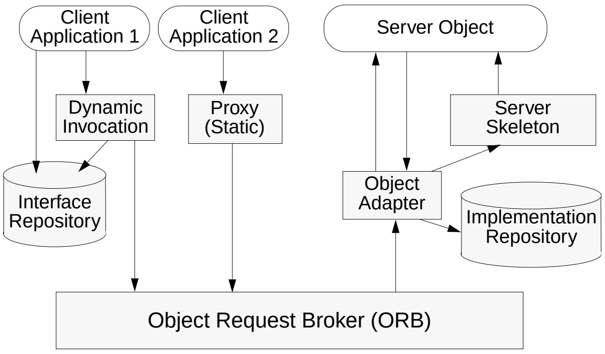
- The *CORBA specification details the interfaces and characteristics of the Object Request Broker*; it practically specifies the middleware functions which allow application objects to communicate with one another no matter where they are located, who has designed them and in which language they are implemented.
- OMG only provides a specification; there are several products which, to a certain extent, implement the OMG specification.

CORBA (cont'd)

- Key concepts:
  - CORBA specifies the middleware services used by the application objects.
  - An object can be a client, a server or both.
  - Object interaction is through requests: the information associated with a request is
    1. an operation to be performed
    2. a target object
    3. zero or more parameters.
  - CORBA supports *static* as well as *dynamic binding*; dynamic binding between objects uses runtime identification of objects and parameters.
  - The *interface* represents the contract between client and server; an IDL has been defined for CORBA; proxies and skeletons (client and server stubs) are generated as result of IDL compilation.
  - CORBA objects do not know the underlying implementation details; an *object adapter* maps the generic model to a specific implementation.

CORBA (cont'd)

Components of a CORBA environment:



Interface and Implementation Repository

Interface Repository

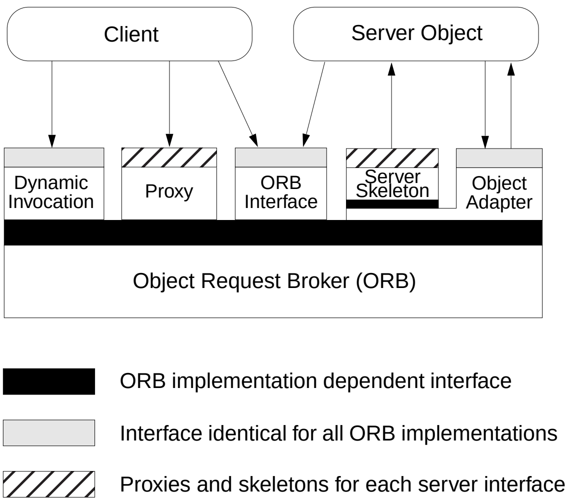
- The interface repository provides a (standard) representation of available object interfaces for all objects in the distributed environment. It corresponds to the server objects' IDL specification.
- The clients can access the interface repository to learn about the server objects, determine the types of operations which can be invoked and the corresponding parameters. This is used for *dynamic invocation* of objects.

Implementation Repository

- Implementation details for the objects implementing each interface are stored in the implementation repository:
  - the main information is a mapping from the server object's name to the file name which implements the respective service;
  - there is information concerning the object methods and information needed for method selection.
- Information stored in the implementation repository can be specific to the operating system running on the respective server object's computer.
- The representation in the implementation repository can be specific for a certain CORBA implementation.
- The implementation repository is used by the *object adapter* in order to solve an incoming call and activate the right object method (via a *server skeleton*).

The Object Request Broker (ORB)

ORB and its interfaces:



The Object Request Broker (cont'd)

☞ The ORB, through its interfaces, provides mechanisms by which objects transparently interact with each other.

- Issuing of a request from a client can be dynamic or static; it is performed through the *proxies (client stubs)* or the *dynamic invocation interface*.
- Invocation of a specific server method is performed by the server skeleton which gets the request forwarded from the *object adapter*.
- The *ORB interface* can be accessed also *directly* by clients and object implementations for certain services: e.g. directory services, services connected to naming, manipulation of object references.

Static and Dynamic Invocation

☞ CORBA allows both *static and dynamic invocation* of objects. The choice is made depending on how much information, concerning the server object, is available at compile time.

Static Invocation

- Static invocation is based on compile time knowledge of the server's interface specification. This specification is formulated in IDL and is compiled into a *proxy (client stub)*, corresponding to the programming language in which the client is encoded.
- For the client, an object invocation is like a local invocation to a proxy method. The invocation is then automatically forwarded to the object implementation through the ORB, the object adapter and the skeleton.
- Static invocation is efficient at run time, because of the relatively low overhead.

Static and Dynamic Invocation (cont'd)

Dynamic Invocation

- Dynamic invocation allows a client to invoke requests on an object without having compile-time knowledge of the object's interface.
- The object and its interface (methods, parameters, types) are detected at run-time. CORBA provides, through the *dynamic invocation interface*, the mechanisms in order to inspect the *interface repository*, to dynamically construct invocations and provide argument values corresponding to the server's interface specification.
- Once the request has been constructed and arguments placed, its invocation has the same effect as a static invocation.
- The execution overhead of a dynamic invocation is huge.
- From the server's point of view, static and dynamic invocation are identical; the server does not know how it has been invoked.
  The server invocation is always issued through its skeleton, generated at compile time from the IDL specification.

The Basic Object Adapter

☞ The object adapter (OA) is the primary interface between the server object implementation and the ORB.

Services provided by the OA:

- Object registration: OA provides operations by which certain entities, specified in a given programming language, are registered as *CORBA objects*.
- Object reference generation: OA generates object references to CORBA objects.
- Object upcalls: OA dispatches incoming requests to the corresponding registered objects.
- Server process and object activation: if needed, OA starts up server processes and activates objects as result of incoming invocations.

Other CORBA Services

These services, and others, have been specified by the CORBA documents; current products implement only some of them.

☞ Naming and Trading Services:
- The basic way an object reference is generated is at 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 objects 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.

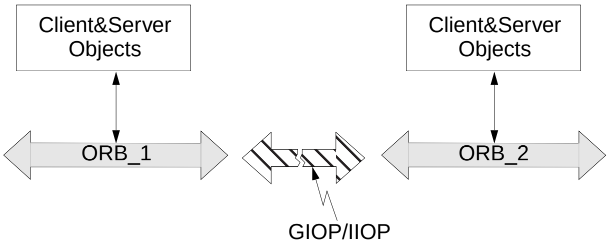
☞ Security Service: protects components from unauthorized users; it provides authentication, access control lists, confidentiality, etc.

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

- -

Inter-ORB Architecture

☞ Implementations of ORBs differ from vendor to vendor ⇒ how do we solve interaction between objects which are running on different CORBA implementations?



- General Inter-ORB Protocol (GIOP): GIOP is defined in CORBA 2.0; it specifies a set of message formats and common data representations for interactions between ORBs and is intended to operate over any connection oriented transport protocol.
- Internet Inter-ORB Protocol (IIOP): IIOP is a particularization of GIOP; it specifies how GIOP messages have to be exchanged over a TCP/IP network.

Summary

- Distributed systems are typically heterogeneous. Middleware is the set of services which enable the components to interact with each other without taking notice of the distributed and heterogeneous character of the environment.
- The API visible for the user of a service is defined in an IDL. The IDL compiler generates proxies and skeletons (client and server stubs). IDLs should be implementation language independent.
- CORBA is the OMGs specification for an Object Request Broker (ORB). Several vendors provide different (partial) implementations consistent with this specification.
- The ORB, through its interfaces, provides mechanisms by which objects transparently interact with each other.
- Objects in CORBA can be invoked statically and dynamically. Static invocation is based on compile time knowledge of the server's interface specification. Dynamic invocation allows a client to invoke requests on an object without having compile-time knowledge of the object's interface.
- The object adapter is the interface between the object implementation and the ORB. It provides services for registration of objects and their activation.
- CORBA 2.0 defines protocols for interaction between ORBs implemented by different vendors.

PEER-TO-PEER SYSTEMS

1. Characteristics of Peer-to-Peer Systems

2. The Napster File System

3. Peer -to-Peer Middleware

Basic Characteristics

☞ Main characteristics of peer-to-peer systems:

- Each user contributes resources to the system.
- All the nodes have the same functional capabilities and responsibilities (although they may differ in the resources they contribute).
- Correct operation does not depend on the existence of any centrally-administered system.

☞ Key issues:

- Choice of strategy for
  - the placement of data and their replica across many hosts;
  - the access to data.

  Such that
  - workload of nodes and communication lines is balanced;
  - availability of data is provided.

☞ Anonymity of providers and users is offered (at least to a certain degree).

Why Do We Need It?

☞ If only particular servers which are centrally managed, can provide services/data, then scalability is limited:

- server capacity
- network bandwidth provided to a server

☞ To avoid the scaling problem

- Peer-to-peer systems use the data and computing resources available in the personal computers and workstations present on the Internet and other networks.
- Instead of separately managed servers, services are provided by all these resources together.

☞ Important!

- Availability of individual processes/computers in a peer-to-peer system is unpredictable

⇩

Services cannot rely on guaranteed access to a host.

- Availability can be improved by replication on several hosts.

The Evolution of Peer-to-Peer Systems

First Generation:
Napster (1999)

☞ The index is centralised!

Second Generation:
Freenet (2000)
Gnutella (2000)
Kazaa (2001)
BitTorrent (2002-2003)

☞ Only semi-centralised or completely distributed.

☞ Better anonymity, scalability, fault tolerance.

Third Generation
Peer-to-Peer Middleware

☞ Platforms for application-independent management of distributed resources.

☞ Used to implement peer-to-peer applications.

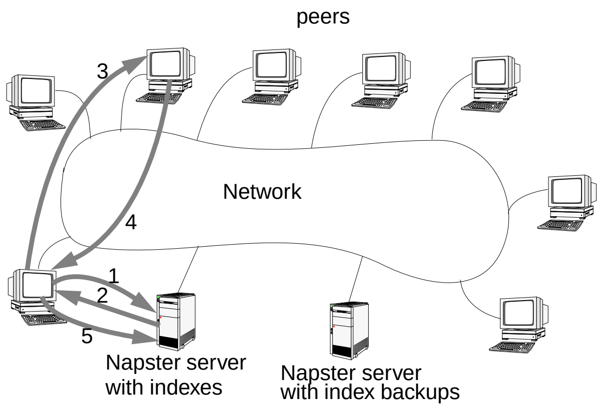
The Napster File Sharing System

☞ Napster provides a globally-scalable information storage and retrieval service for digital music files.

☞ Napster was the first to demonstrate the feasibility of a peer-to-peer solution on large scale.

☞ Napster, as an open service, was shut down July 2001, as result of lawsuits on copyright issues.

The Napster File Sharing System (cont'd)



Step 1: File location request;
Step 2: List of peers offering the files;
Step 3: File request;
Step 4: File loading;
Step 5: Index update (user adds own files to pool of shared resources).

The Napster File Sharing System (cont'd)

☞ Napster uses a centralised index (with replicas for increased availability).

☞ The whole pool of files is distributed over the personal computers of the peers.

☞ In order to achieve load balancing:

- When creating and sending the list of peers offering the file (step 2), Napster takes into account locality (the distance between the requesting client and the potential servers).

Problems with Napster

☞ Centralised index:

- Scaling problem (server capacity and network bandwidth).
- Anonymity of operators is not possible: for example, legal responsibility for copyright issues can be put on operators maintaining the central index.

☞ A completely distributed index can both provide better scaling and anonymity.

☞ Napster did not provide particular solutions for consistency of replica updates or for guaranteed availability. This was no problem because of the particular application, music files:

- Music files are immutable (they don't change after being created) ⇒ there is no need to maintain replicas consistent.
- If a file is unavailable at a certain moment it can be downloaded later.

☞ Second generation systems (see slide 26) have tried to solve the above problems by applying various specific, ad hoc solutions.

Peer-to-Peer Middleware

- Peer-to-peer middleware systems provide a support for the implementation of distributed services that are located across many hosts in a widely distributed network.

- They provide a programming interface to application programmers for the implementation of peer-to-peer applications.

- No completely mature commercial products yet available.

Peer-to-Peer Middleware (cont'd)

☞ The main function of peer-to-peer middleware:

- automatic placement, replication and subsequent location of distributed objects managed by the peer-to-peer systems.

☞ Functionality supported:

- add and remove hosts to/from the system;
- add and remove resources (objects) to/from the systems;
- allow clients to locate any individual resource made available and communicate with it.

Requirements for Peer-to-Peer Middleware

☞ Scalability

- Exploit the hardware resources of a very large number of hosts.
- Support applications with millions of objects located on tens/hundreds of thousands of hosts.

☞ Load balancing: balanced distribution of workload among computers and network links.

- Random placement of resources.
- Use of replicas for heavily-used resources.

☞ Optimization of interaction: the distance between nodes that interact affects the response time and the load of the network.

- Resources should be placed close to nodes that access them the most.

Requirements for Peer-to-Peer Middleware (cont'd)

☞ Adaptation to dynamic host availability: computers can whenever join the system or leave it.

- Provide a dependable service despite the unpredictable availability of the infrastructure.
- When a host joins the system the resources provided are integrated and the load redistributed.
- When a host leaves the system, the load and resources are redistributed.
- Systematic replication of objects for availability

☞ Security

- Proper authentication and encryption mechanisms to ensure integrity and privacy.

☞ Anonymity, deniability, resistance to censorship

- Keep anonymity of holders and recipients of data;
- Plausible denial of responsibility for holding/ supplying data.

Summary

- Peer-to peer systems are a possible solution for the scaling problems with traditional client-server systems.
- Scaling in peer-to-peer systems is solved by exploiting the resources available on the personal computers and workstations available in the network, instead of using dedicated and centrally maintained servers.
- Napster has been the first widely used peer-to-peer system. While the pool of files is completely distributed over the personal computers of the hosts, Napster is still using a centralised index. This has consequences with regard to both scaling and anonymity.
- Peer-to-peer middleware is supporting the implementation of peer-to-peer applications. They provide automatic placement, replication, and location of objects in the peer-to-peer systems.
- Peer-to-peer systems are highly efficient to store and manage very large amounts of objects which are immutable (they don't change after being created) or which only rarely are updated.