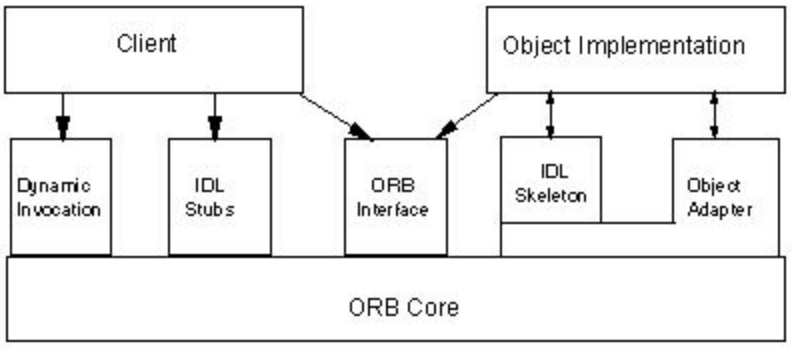
CORBA

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, 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 invoke 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 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 have a public interface which is used by the object implementation and a private interface that is used by the IDL skeleton.

Source :- http://www.nyu.edu/classes/jcf/g22.3033-011_fa01/handouts/g22_3033_011_h41.htm

Jini

Jini is a service-oriented architecture that defines a programming model that both exploits and extends Java technology. This programming model enables the construction of secure, distributed systems consisting of federations of well-behaved network services. Jini helps to build networks that are scalable and flexible, which are required attributes in distributed computing scenarios.

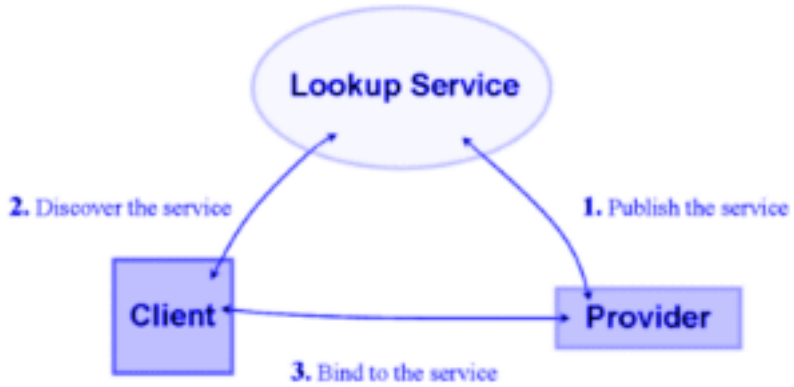
Jini's main objective is to shift the focus of distributed computing from a disk-drive-oriented approach to a network-adaptive approach by developing scalable, evolvable and flexible dynamic computing environments. Jini makes resources over a network look like local resources.

Using Jini, users will be able to plug printers, storage devices, speakers, and any kind of device directly into a network and every other computer, device, and user on the network will know that the new device has been added and is available. Each pluggable device will define itself immediately to a network device registry. When someone wants to use or access the resource, their computer will be able to download the necessary programming from it to communicate with it. No longer will the special device support software known as a device driver need to be present in an operating system. The operating system will know about all accessible devices through the network registry.

Jini not only allows for the addition of printers, storage and other devices to a network, it also allows the devices to be detected automatically over the network without having to reboot the system. Hardware devices declare to their own operating systems as well as to other computers, devices and users on the network that they have been added and are available for use. This is possible because the devices define themselves to a network device registry soon after they have been added.

Jini consists of four program layers:

- Directory Service
- JavaSpace
- Remote Method Invocation (RMI)
- Boot, Join, and Discover Protocol



The Jini architecture is divided into three main parts:

- Client: The user who accesses the resources shared over a network
- Server: The system to which the resources are attached
- Lookup Service: Services for resources such as printers, storage devices and speakers, which are attached to the server and made available to clients over the network

Jini has the following key advantages:

- Provides stable networking solutions
- Helps in upgrading systems
- Helps to keep old clients running while adding new ones
- Helps build scalable, dynamic and flexible networks

Limitations

Jini uses a lookup service to broker communication between the client and service. This appears to be a centralized model (though the communication between client and service can be seen as decentralized) that does not scale well to very large systems. However, the lookup service can be horizontally scaled by running multiple instances that listen to the same multicast group.

Source: https://docs.gigaspaces.com/xap/12.2/overview/about-jini.html

Mach

Mach is a kernel developed at Carnegie Mellon University to support operating system research, primarily distributed and parallel computing. Mach is often mentioned as one of the earliest examples of a microkernel. However, not all versions of Mach are microkernels.

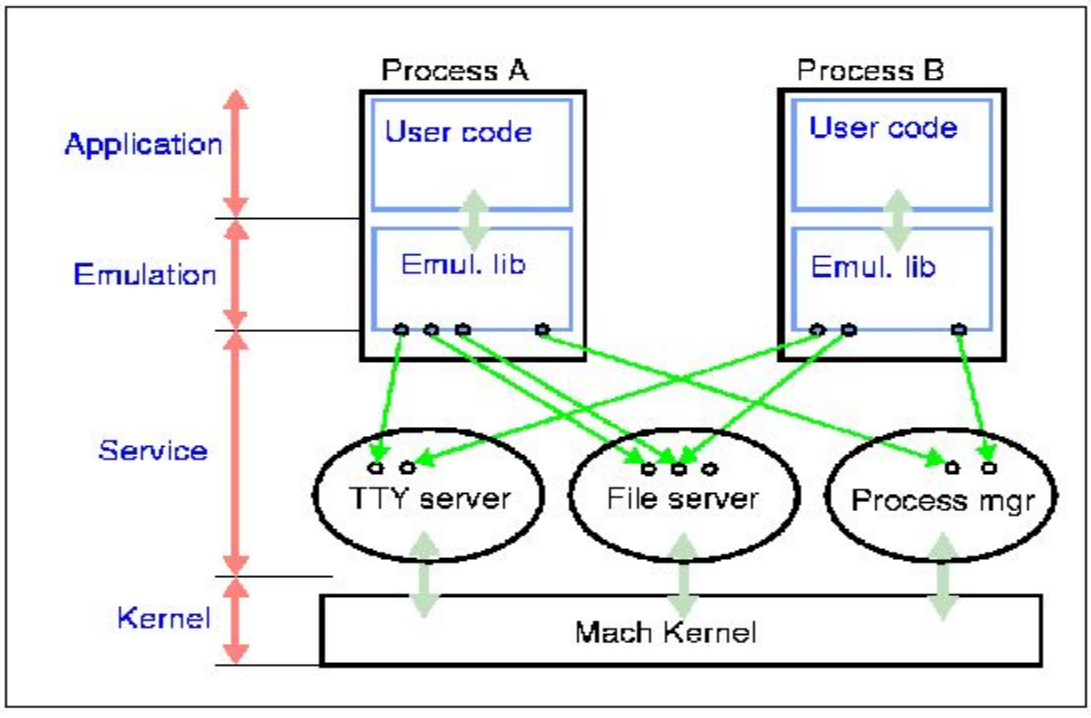
Mach was developed as a replacement for the kernel in the BSD version of Unix, so no new operating system would have to be designed around it. Mach and its derivatives exist within a number of commercial operating systems. These include all using the XNU operating system kernel which incorporates an earlier non-microkernel Mach as a major component.

The Mach operating system was designed with the following three critical goals in mind:

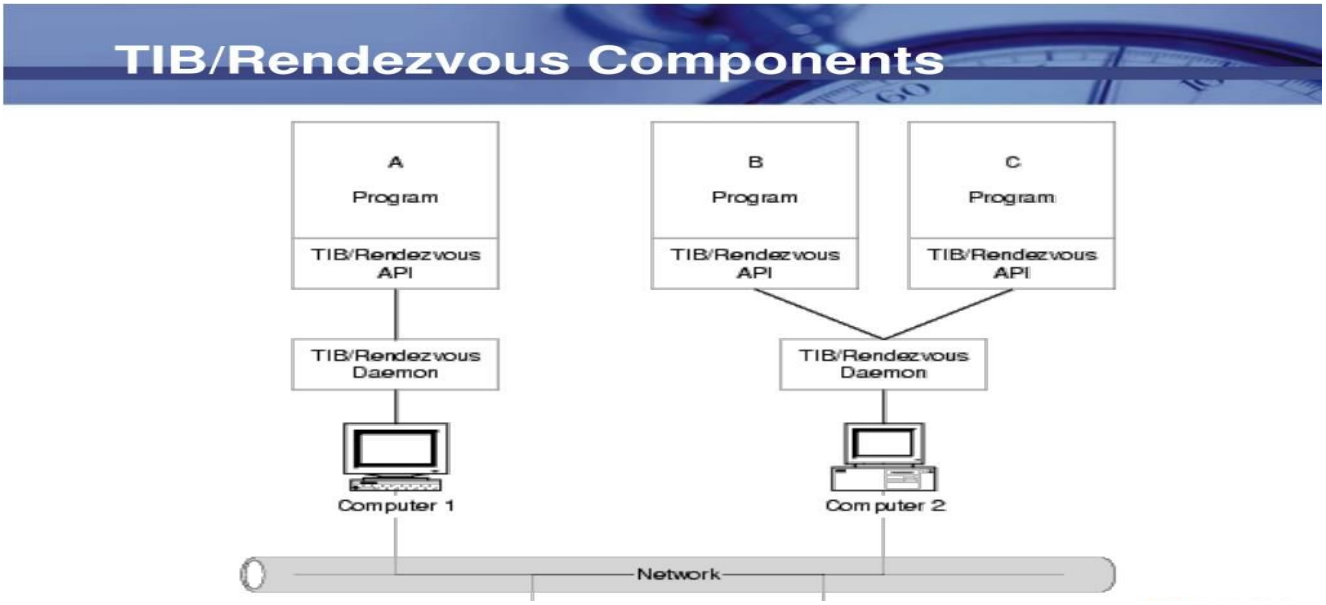
1. Emulate 4.3BSD UNIX so that the executable files from a UNIX system can run correctly under Mach.

2. Be a modern operating system that supports many memory models, as well as parallel and distributed computing.

3. Have a kernel that is simpler and easier to modify than is 4.3BSD. Mach's development followed an evolutionary path from BSD UNIX systems. Mach code was initially developed inside the 4.2BSD kernel, with BSD kernel components replaced by Mach components as the Mach components were completed.



TIB / Rendezvous



TIB/Rendezvous makes it easy to create distributed applications across heterogeneous systems. In coordination-based systems where published data items are matched only against live subscribers, reliable communication plays a crucial role. In this case, fault tolerance is most often implemented through the implementation of reliable multicast systems that underly the actual publish/subscribe software. There are several issues that are generally taken care of. First, independent of the way that content-based routing takes place, a reliable multicast channel is set up. Second, process fault tolerance needs to be handled.

TIB/Rendezvous assumes that the communication facilities of the underlying network are inherently unreliable. To compensate for this unreliability, whenever a rendezvous daemon publishes a message to other daemons, it will keep that message for at least 60 seconds. When publishing a message, a daemon attaches a (subject independent) sequence number to that message. A receiving daemon can detect it is missing a message by looking at sequence numbers (recall that messages are delivered to all daemons). When a message has been missed, the publishing daemon is requested to retransmit the message.

Much of the reliability of communication in TIB/Rendezvous is based on the reliability offered by the underlying network. TIB/Rendezvous also provides reliable multicasting using (unreliable) IP multicasting as its underlying communication means. The scheme

followed in TIB/Rendezvous is a transport-level multicast protocol known as Pragmatic General Multicast (PGM).

Implementation of TIB

- One of the two communicating tasks knows the name of the other and names it explicitly.

- The second task knows only that it expects some external interaction.

- It is based on subject based addressing.

- Receiving a message on subject X is possible only if receiver had subscribed to X.

- Publishing a message on subject X means sending message to all subscribers of X.

Source:- http://csis.pace.edu/~marchese/CS865/Lectures/Chap13/Chapter13.htm