Chapter One

1. Integrative Programming Fundamentals

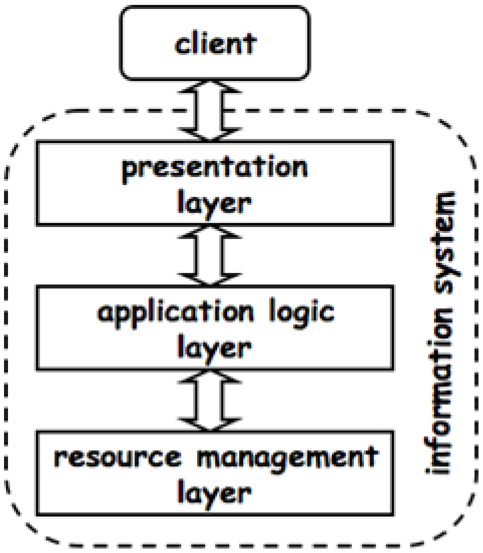
1.1. Distributed Information Systems

Distributed computing is a system whereby the code to run an application is spread across multiple computers. For example, to create a large transaction processing system, you might have a separate server for business logic objects, one for presentation logic objects, a database server, and so on, all of which need to talk to each other. It is a system where parts of an application/system run on multiple computers simultaneously.

Information System Design

- ✓ Layers and Tiers
- ✓ Bottom Up Design
- ✓ Top Down Design

Layers and Tiers

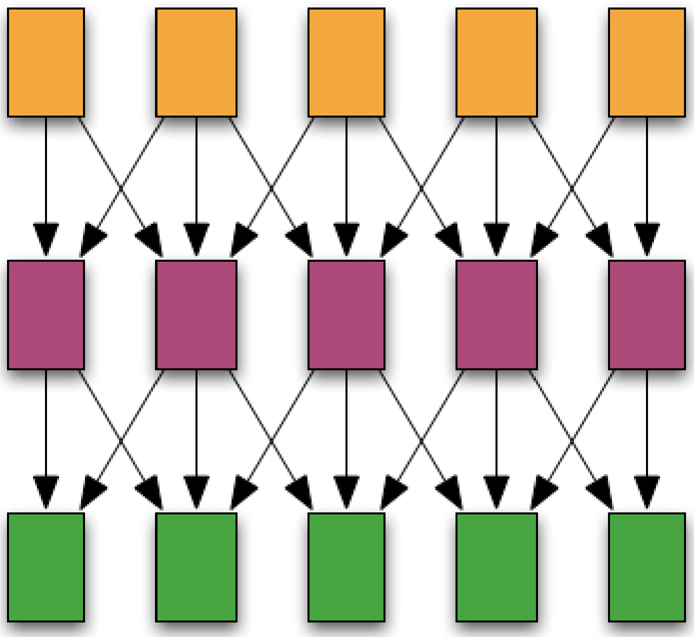


A client is any user or program that wants to perform an operation on the system. Clients interact with the system through a presentation layer.

The application logic determines what the system actually does. It takes care of enforcing the business rules and establishing the business process. The application logic can take many forms: programs, constraints, workflows, etc.

The resource manager deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a database but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence.

Boxes and Arrows

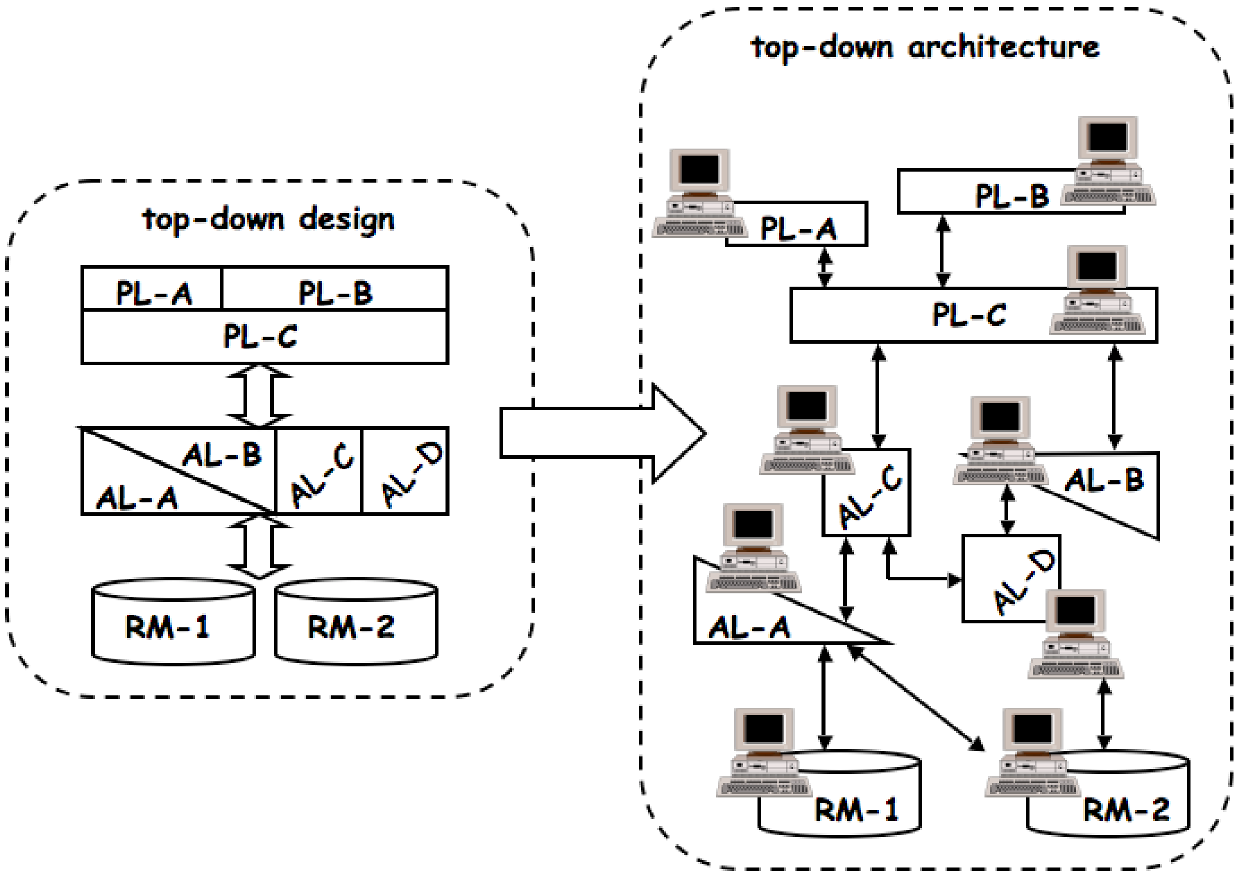


Each box represents a part of the system. Each arrow represents a connection between two parts of the system.

Adding boxes makes the system modular: this provides opportunities for adding distribution and parallelism. It also supports encapsulation, component–based design, reuse, etc. Adding arrows, on the other hand, adds connections that need to be maintained; more coordination is necessary. The system becomes more complex to monitor and manage.

The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to data. Performance suffers considerably. System designers try to balance the flexibility of modular design with the performance demands of real applications.

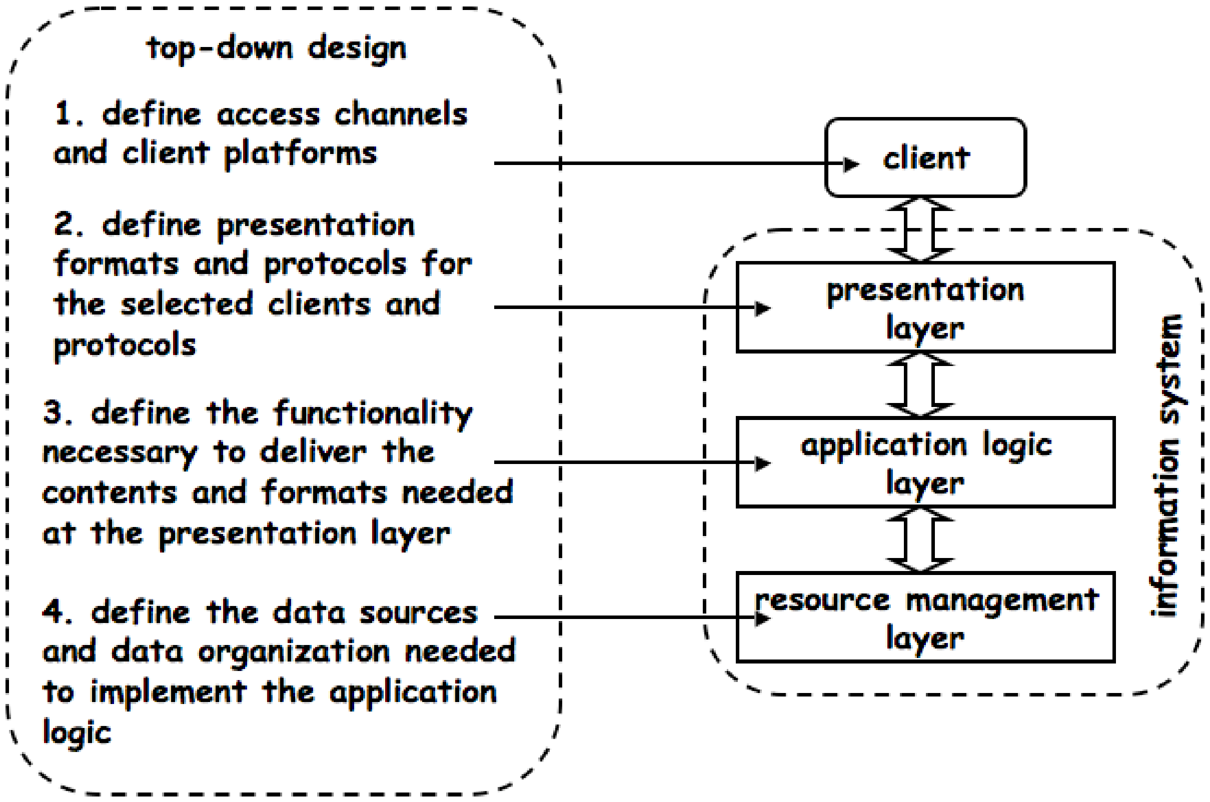
Top down Design



The functionality of a system is divided among several modules. Modules are typically not stand-alone components; their functionality depends on modules located in a lower layer.

Hardware is typically homogeneous and the system is designed to be distributed from the beginning.

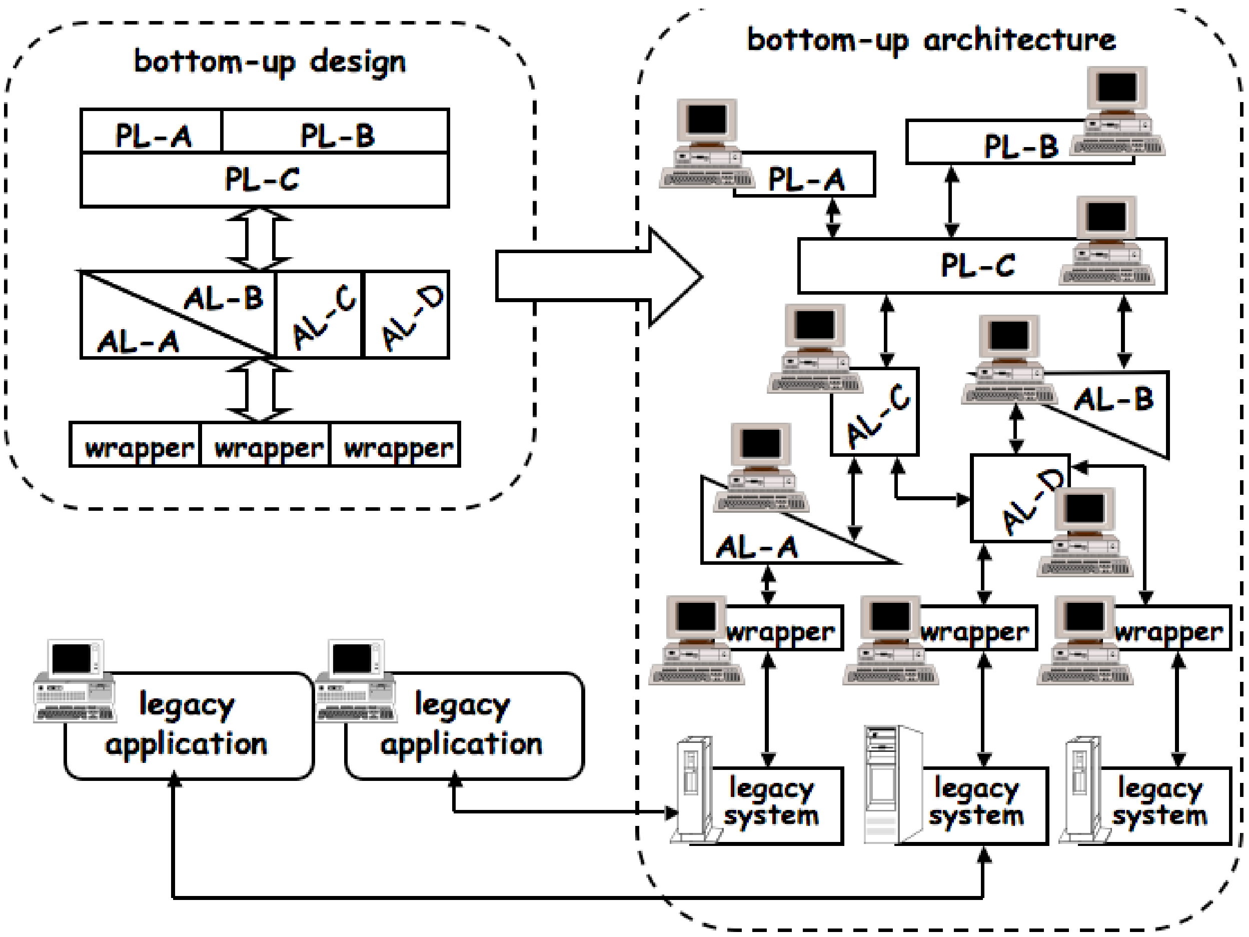
The process of top down design


top-down design
1. define access channels and client platforms
2. define presentation formats and protocols for the selected clients and protocols
3. define the functionality necessary to deliver the contents and formats needed at the presentation layer
4. define the data sources and data organization needed to implement the application logic
client
presentation layer
application logic layer
resource management layer
information system

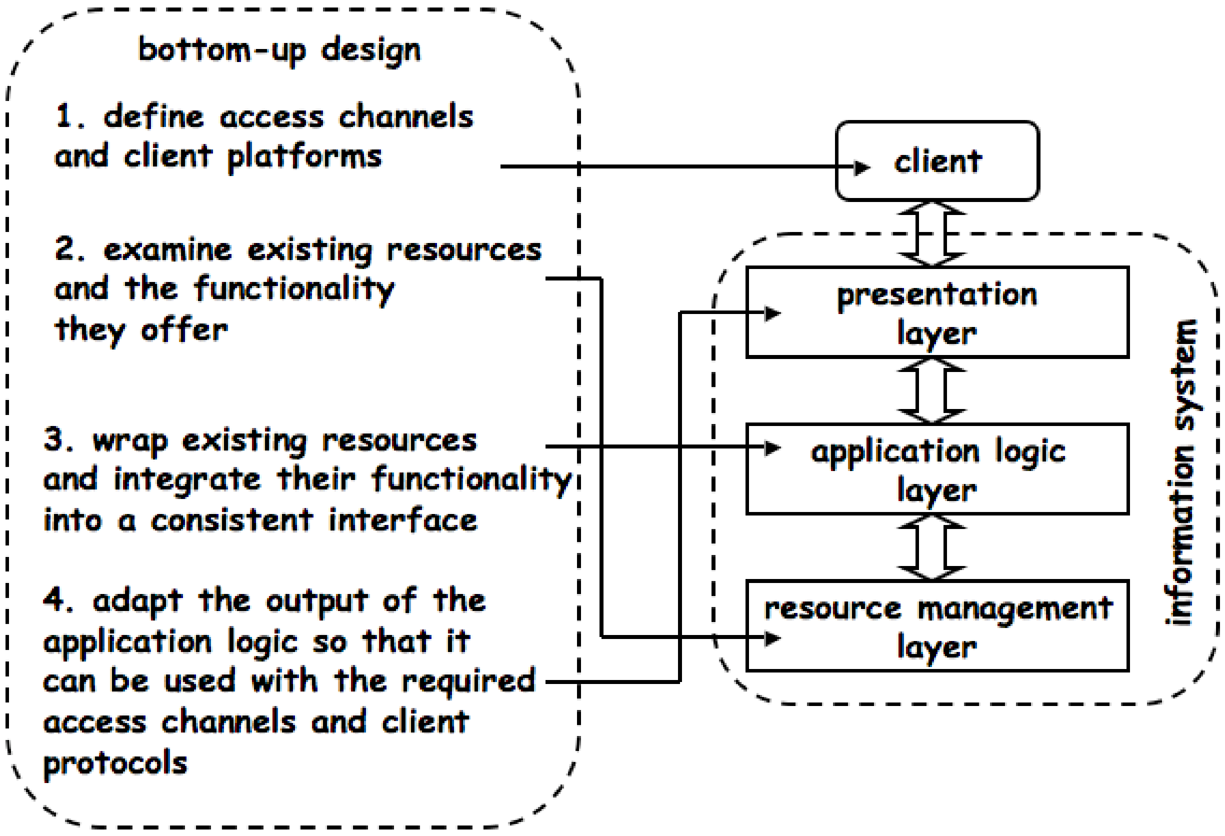
Bottom up Design

- ✓ In a bottom up design, many of the basic components already exist. These are stand-alone systems which need to be integrated into a new system.
- ✓ The components do not necessarily cease to work as stand-alone components. Often old applications continue running at the same time as new applications.
- ✓ This approach is used widely because legacy systems exist and typically cannot be easily replaced.

- ✓ Much of the work and products in this area are related to middleware, the intermediate layer used to provide a common interface, bridge heterogeneity, and cope with distribution.



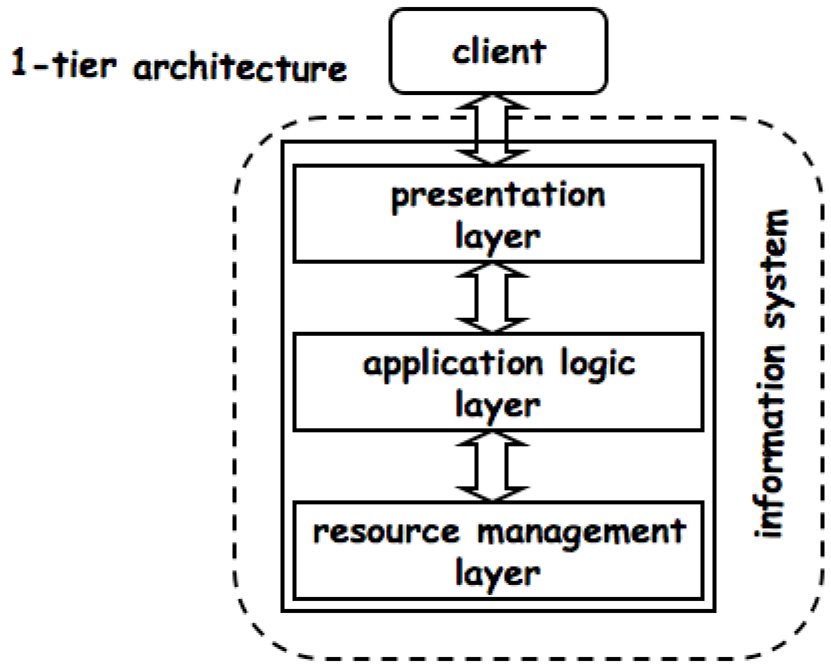
The process of Bottom up Design


bottom-up design
1. define access channels and client platforms
2. examine existing resources and the functionality they offer
3. wrap existing resources and integrate their functionality into a consistent interface
4. adapt the output of the application logic so that it can be used with the required access channels and client protocols
client
presentation layer
application logic layer
resource management layer
information system

Information System Architecture

- ✓ One Tier
- ✓ Two Tier (client/server)
- ✓ Three Tier (middleware)
- ✓ N-tier Architectures
- ✓ Clusters and Tier Distribution

One Tier: Monolithic Systems



The presentation layer, application logic and resource manager are built as a monolithic entity.

Users/programs access the system through “dumb” terminals, whose display is controlled by the information system.

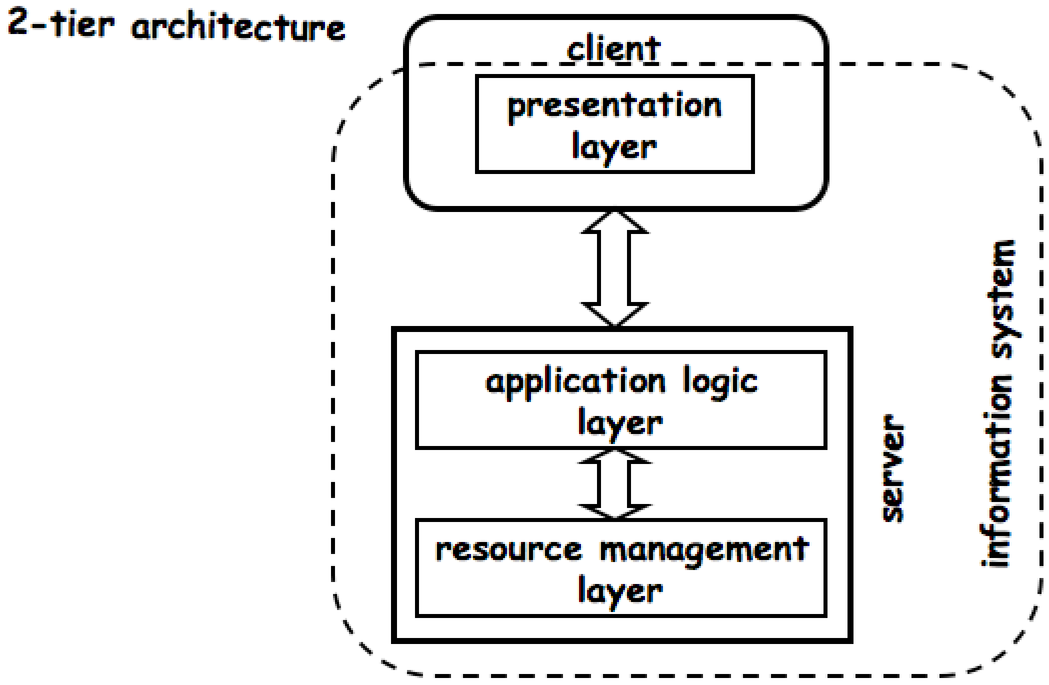
This was the typical architecture of mainframes, offering several advantages:

- ✓ no forced context switches in the control flow
- ✓ everything is centralized; managing and controlling resources is easier
- ✓ the design can be highly optimized by blurring the separation between layers

Two Tier: Client/Server

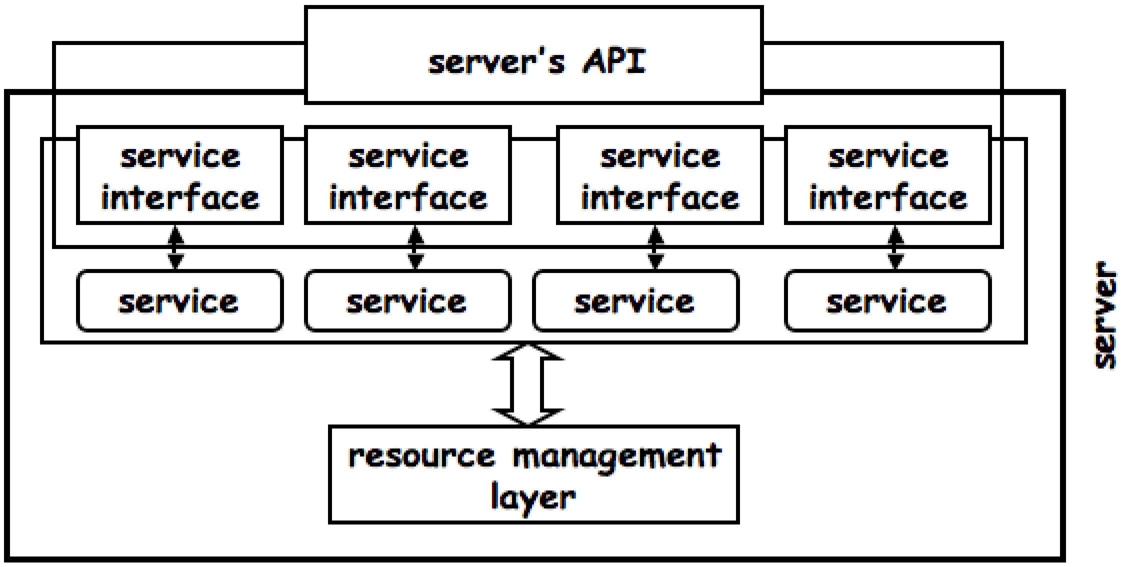
- ✓ As computers became more powerful, it was possible to move the presentation layer to the client. This has several advantages:
- ✓ Clients are independent of each other: one can have several presentation layers depending on what each client needs to do.

- ✓ One can take advantage of the computing power at the client machine to have more sophisticated presentation layers while also saving computer resources on the server.
- ✓ It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside.
- ✓ The resource manager only sees one client: the application logic. This greatly helps with performance since there are no client connections/sessions to maintain.



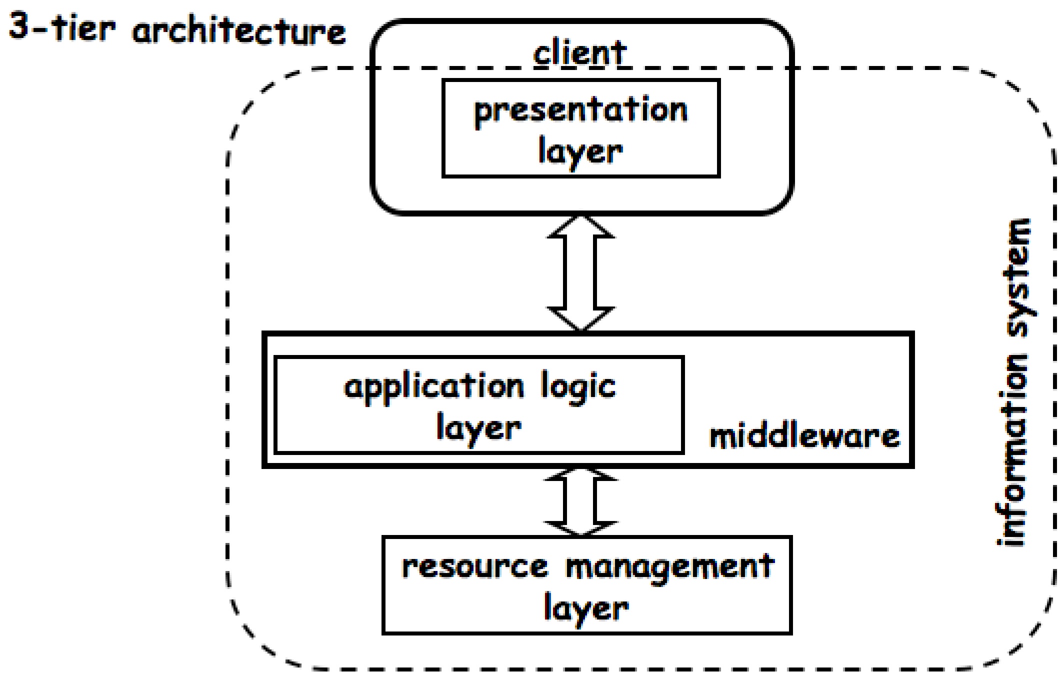
Two Tier: Server API

- ✓ Client/server systems introduced the notion of service (the client invokes a service implemented by the server)
- ✓ Client/server systems also introduced the notion of service interface (how the client can invoke a given service)
- ✓ Taken together, the interfaces to all the services provided by a server define the server's API

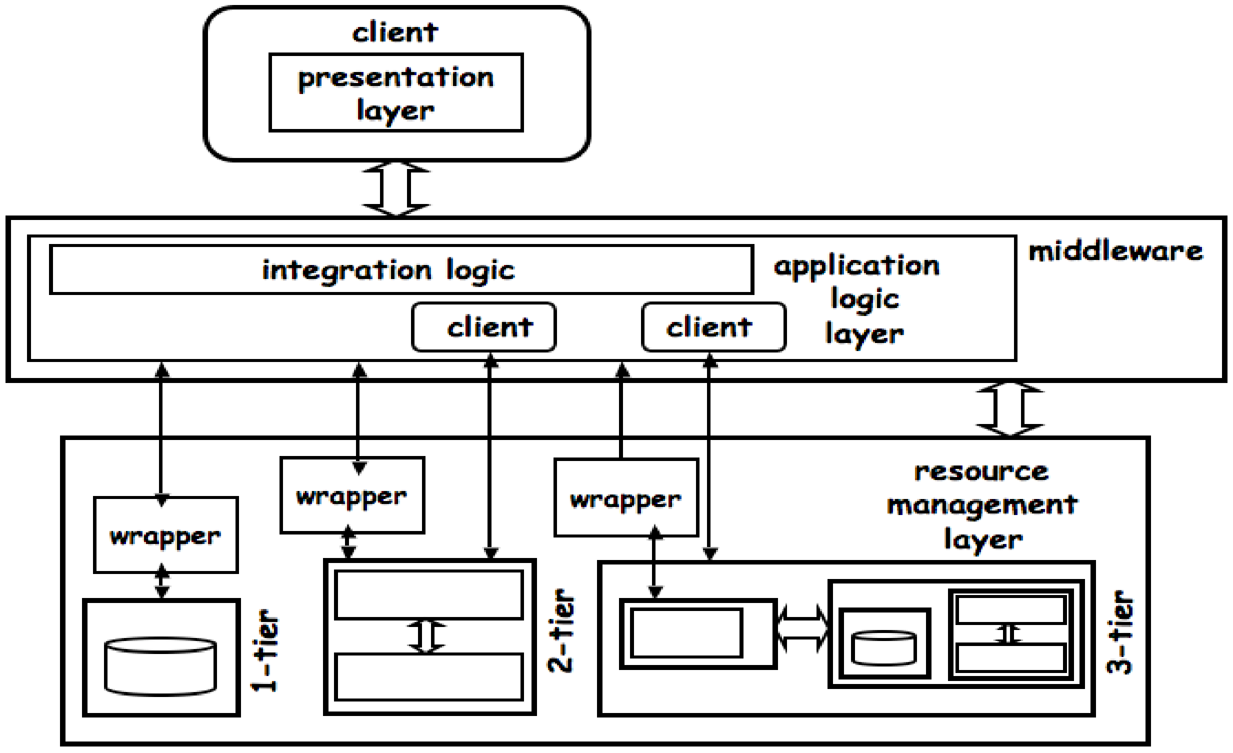


Three Tier: Middleware

- ✓ In a 3 tier system, the three layers are fully separated; they are also typically distributed
- ✓ Middleware introduces an additional layer of business logic encompassing all underlying systems
- ✓ By doing this, a middleware system:
  - ❖ simplifies the design of clients by reducing the number of interfaces it needs to know
  - ❖ provides transparent access to the underlying systems
  - ❖ acts as a platform for inter-system functionality and high level application logic
  - ❖ takes care of locating resources, accessing them, and gathering results

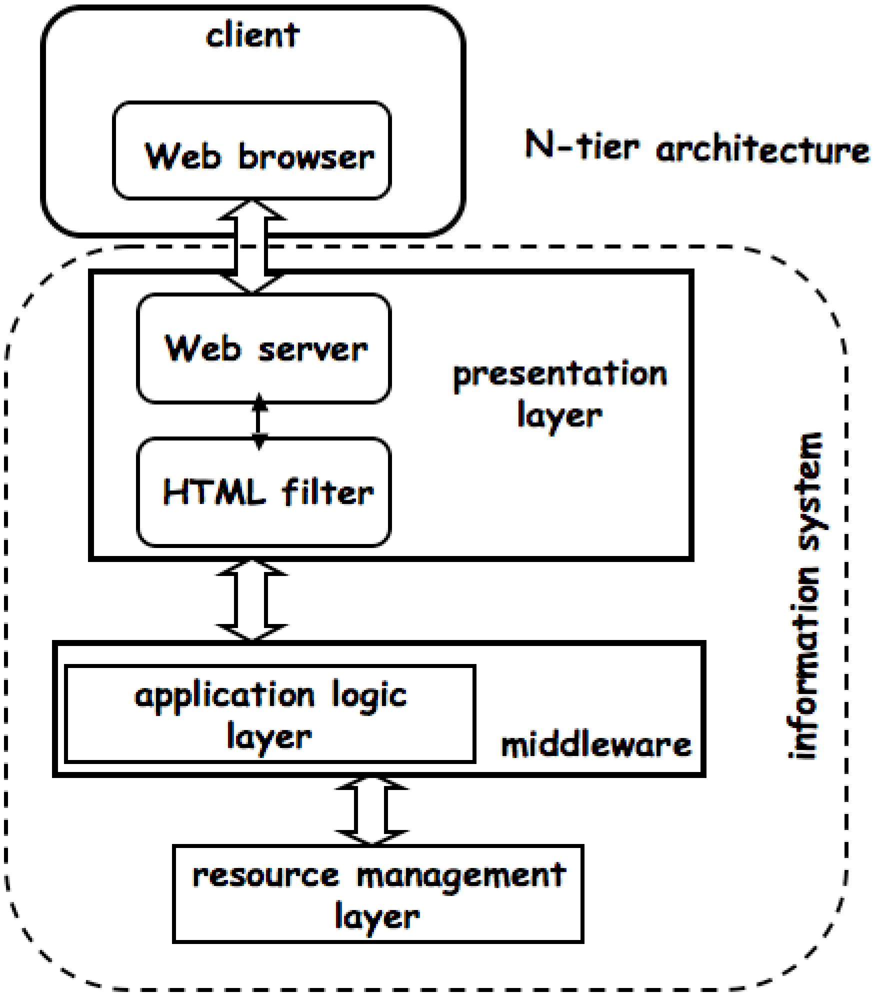


Middleware systems also enable the integration of systems built using other architectures

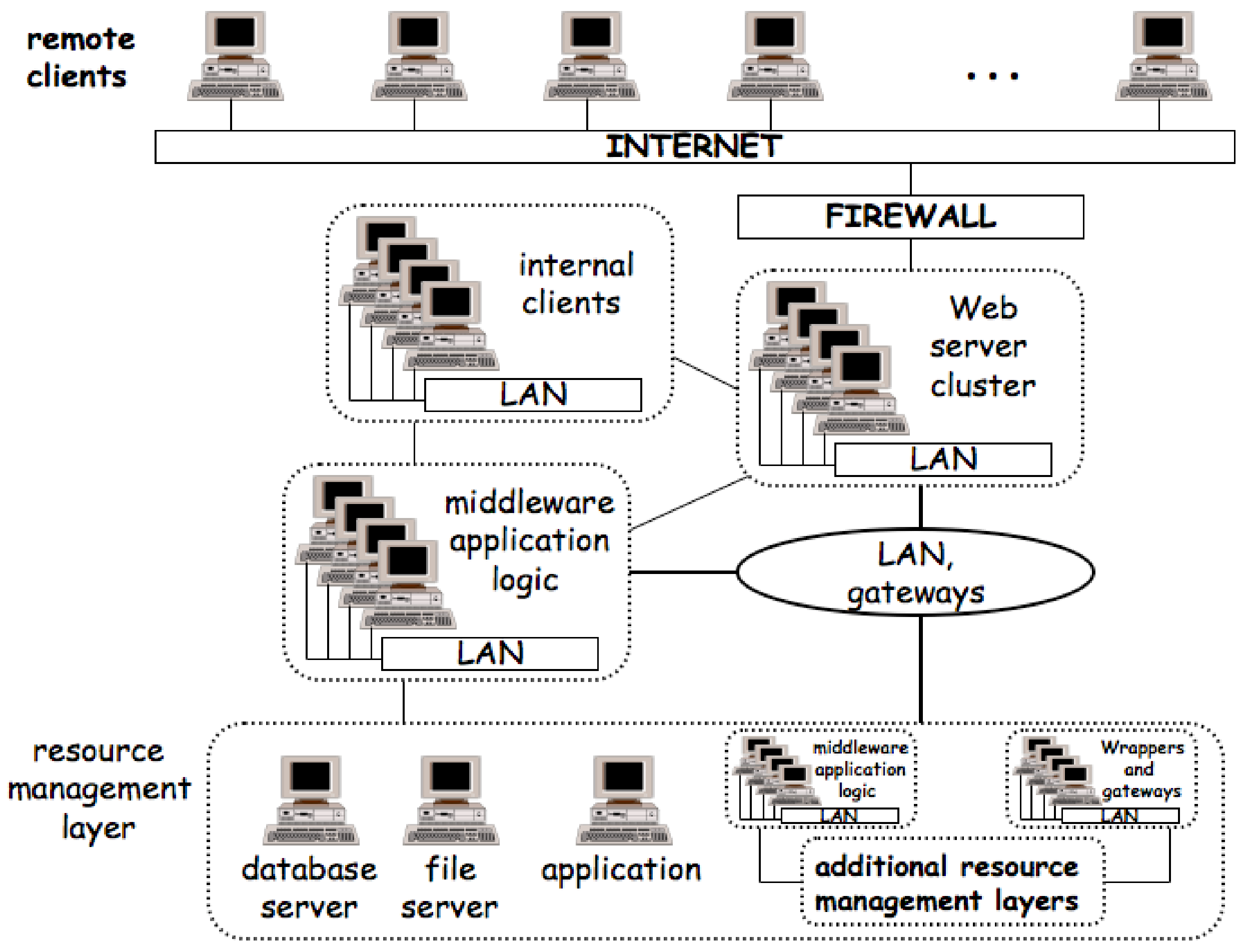


N-Tier: Web Integration

- ✓ N-tier architectures result from connecting several 3-tier systems to each other and/or by adding an additional layer to allow clients to access the system via the Web
- ✓ The Web layer was initially external to the information system (a true additional layer); today, it is being incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure in a three tier system, or part of the server directly in a two tier system)
- ✓ The addition of the Web layer led to the notion of “application servers” which was used to refer to middleware platforms supporting Web access



N-Tier Systems in the real world



1.2. Middleware Technologies

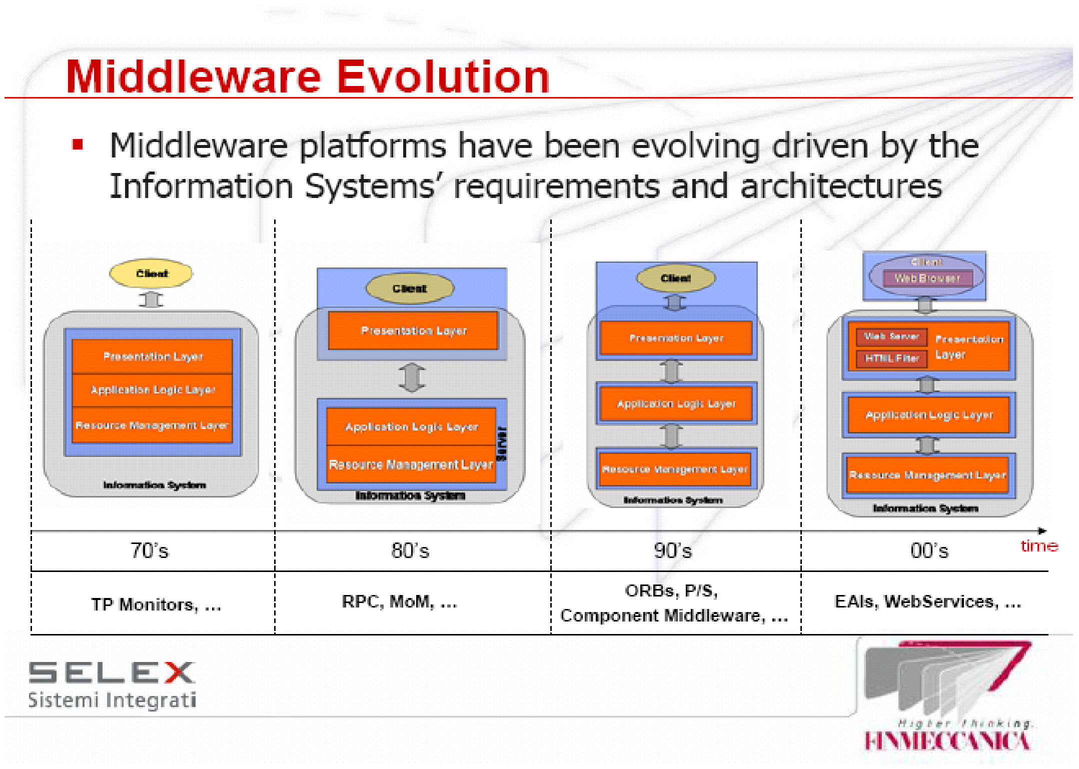
What is middleware?

- ☐ Connectivity software that provides a mechanism for processes to interact with other processes running on multiple networked machines.
- ☐ Facilitates and manages the interaction between applications across heterogeneous computing platforms.
- ☐ Provides programming abstractions thereby reduces hardware, software, network and other technical complexities to the client.

- ☐ Middleware can be seen as a set of programming abstractions that make it easier to develop complex distributed systems
- ☐ To understand a particular class of middleware, you need to understand its model and its API

Brief History of EAI/Middleware

- ☐ Enterprise applications, from as early as the 1960s through the late 1970s, were simple in design and functionality, developed largely in part to end repetitive tasks.
- ☐ By the 1980s, several corporations were beginning to understand the value and necessity for application integration.
- ☐ Enterprise Resource Planning (ERP)
- ☐ As ERP applications became much more prevalent in the 1990s, there was a need for corporations to be able to leverage already existing applications and data within the ERP system; this could only be done by introducing EAI.



Middleware Evolution

- Middleware platforms have been evolving driven by the Information Systems' requirements and architectures

Types of Middleware

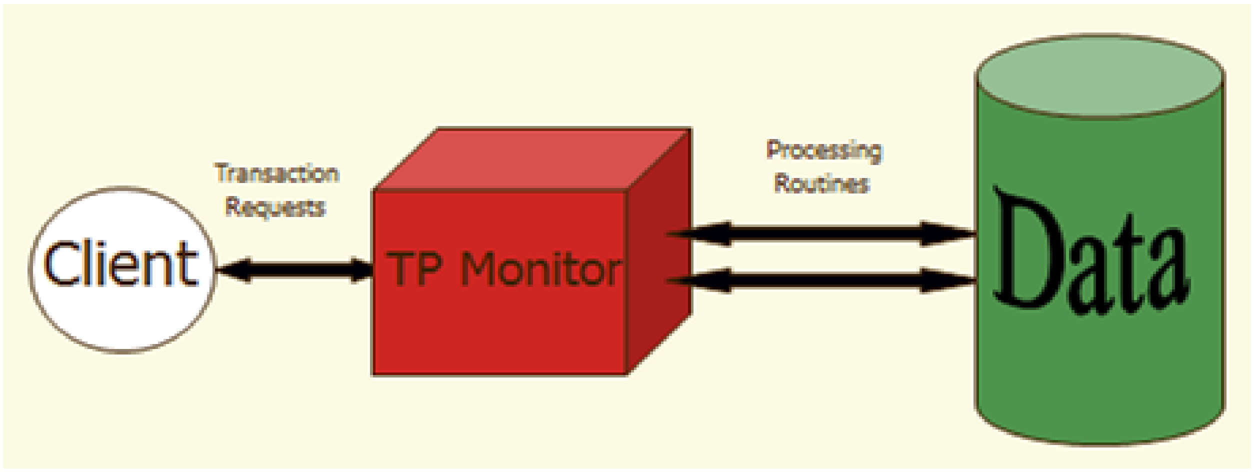
- ☐ There are four basic types of middleware:
  - ■ Transaction Processing Monitor (TP Monitor)
  - ■ Remote Procedure Call (RPC)
  - ■ Message Oriented Middleware (MOM)
  - ■ Object Request Broker (ORB)

Transaction Processing Monitor (TP)

- ☐ Provide a mechanism to facilitate the communication between two or more applications
- ☐ As well as a location for application logic
- ☐ Are based on the concept of a transaction
- ☐ Manages transactions in large database systems
- ☐ Acting like an operating system for transaction processing
- ☐ A program that monitors transactions
- ☐

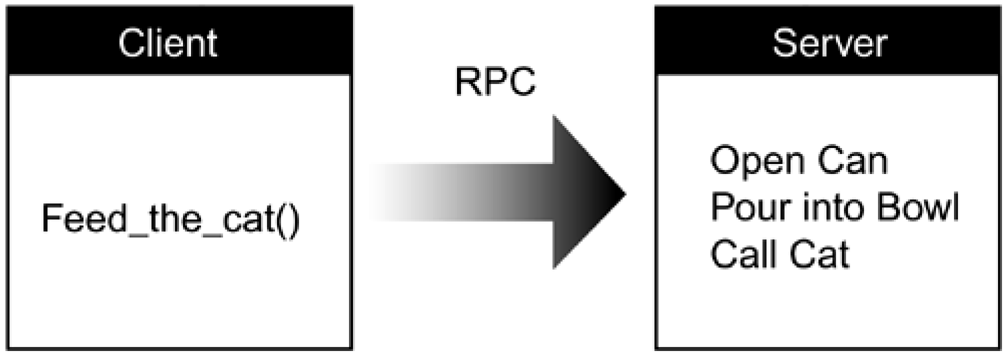
- TP Can provide the following:
  - Manage transactions from their point of origins
  - Ensure ACID property of databases
  - Performs a transaction service that has ability to execute transactions concurrently while at the same time ensuring integrity
  - •Update multiple database systems in a single transaction
  - •Connectivity to a variety of data sources including flat files, non-relational databases etc

TP architecture



Remote Procedure Call (RPC)

- RPC is a client/server mechanism that allows the program to be distributed across multiple platforms.
- Oldest type of middleware
- Are also the easiest to understand and use
- Provide developers with the ability to invoke a function within one program and have that function execute within another program on a remote machine.
- To the developer, the function executes locally. The fact that it is actually carried out on a remote computer is hidden.
- RPCs are synchronous. In order to carry out an RPC, the RPC must stop the execution of the program.



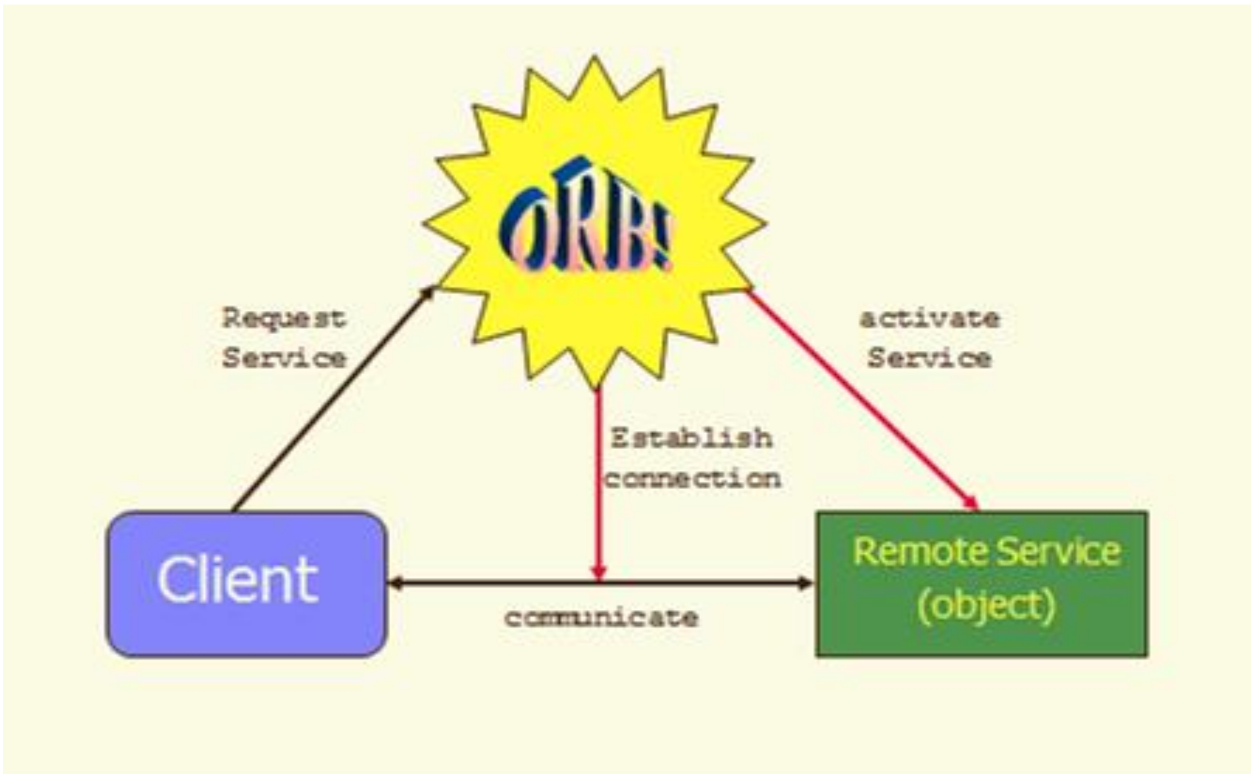
Message Oriented Middleware (MOM)

- Message Oriented Middleware
  - Functionality similar to RPC
  - Provides asynchronous communication between client and server apps by queuing messages temporarily when either of both sides busy.
- RPCs shortcomings led to the creation of MoM,
  - ***to address those shortcomings by the use of messages***
- Uses message (byte-sized units) as a mechanism to move information from point to point
- Because of the notion of messaging, direct coupling is not required.
- MoM relies on asynchronous paradigm (i.e. asynchronous version of RPC)

Message Request Broker (ORB)

- DOs are small programs that use standard interfaces and protocols to communicate an object with another object
- ORB Overcome:
  - No boundaries regarding machine, software and vendors
- Major functionality includes
  - Interface definition
  - Location and activation of remote objects
  - Communication between clients and objects

ORB Architecture



The Different Middleware Protocols

Several protocols exist for performing remote procedure calls, but the most common are DCOM (Distributed Component Object Model) and IIOP (Internet Inter-ORB Protocol), both of which are extensions of other technologies: COM and CORBA, respectively, and Java RMI. Each of these protocols provides the functionality needed to perform remote procedure calls, although each has its drawbacks. The following sections discuss these protocols and those drawbacks, without providing a lot of technical details.

DCOM

Microsoft developed a technology called the Component Object Model, or COM (see www.microsoft .com/com/default.asp), to help facilitate component-based software, software that can be broken down into smaller, separate components, which can then be shared across an application, or even across multiple applications. COM provides a standard way of writing objects so they can be discovered at runtime and used by any application running on the computer. In addition, COM objects are language independent. That means you can write a

COM object in virtually any programming language—C, C++, Visual Basic, and so on—and that object can talk to any other COM object, even if it was written in a different language.

A good example of COM in action is Microsoft Office: Because much of Office's functionality is provided through COM objects, it is easy for one Office application to make use of another. For example, because Excel's functionality is exposed through COM objects, you might create a Word document that contains an embedded Excel spreadsheet.

However, this functionality is not limited to Office applications; you could also write your own application that makes use of Excel's functionality to perform complex calculations, or uses Word's spellchecking component. This enables you to write your applications faster, as you don't have to write the functionality for a spell-checking component or a complex math component yourself. By extension, you could also write your own shareable components for use in others' applications.

COM is a handy technology to use when creating reusable components, but it doesn't tackle the problem of distributed applications. In order for your application to make use of a COM object, that object must reside on the same computer as your application. For this reason, Microsoft developed a technology called Distributed COM, or DCOM. DCOM extends the COM programming model, enabling applications to call COM objects that reside on remote computers. To an application, calling a remote object from a server using DCOM is just as easy as calling a local object on the same PC using COM—as long as the necessary configuration has been done ahead of time.

Nonetheless, as handy as COM and DCOM are for writing component-based software and distributed applications, they have one major drawback: Both of these technologies are Microsoft-specific. The COM objects you write, or that you want to use, will only work on computers running Microsoft Windows; and even though you can call remote objects over DCOM, those objects also must be running on computers using Microsoft Windows.

IIOP

Prior even to Microsoft's work on COM, the Object Management Group, or OMG (see www.omg.org/) developed a technology to solve the same problems that COM and DCOM try to solve, but in a platform- neutral way. They called this technology the Common Object Request Broker Architecture, or CORBA (see www.corba.org/). As with COM, CORBA objects can be written in virtually any programming language, and any CORBA object can talk to any other, even if it was written in a different language. CORBA works similarly to COM, the main difference being who supplies the underlying architecture for the technology.

Although the concepts are the same, using an ORB instead of the operating system to provide the base object services offers one important advantage: It makes CORBA platform independent. Any vendor that creates an ORB can create versions for Windows, Unix, Linux, and so on.

Furthermore, the OMG created the Internet Inter-ORB Protocol (IIOP), which enables communication between different ORBs. This means that you not only have platform independence, you also have ORB independence. You can combine ORBs from different vendors and have remote objects talking to each other over IIOP (as long as you avoid any vendor-specific extensions to IIOP).

Neither COM nor CORBA are easy to work with, which dramatically reduced their acceptance and take up. Although COM classes are reasonably easy to use, and were the basis of thousands of applications including Microsoft Office, they are difficult to design and create. CORBA suffered similar problems, and these difficulties, as well as such scenarios as DLL hell in COM (mismatched incompatible versions of libraries of a machine) led to the design of other techniques.

Java RMI

Both DCOM and IIOP provide similar functionality: a language-independent way to call objects that reside on remote computers. IIOP goes a step further than DCOM, enabling components to run on different platforms. However, a language already exists that is specifically designed to enable you to "write once, run anywhere": Java.

Java provides the Remote Method Invocation, or RMI, system (see http://java.sun.com/products/ jdk/rmi/) for distributed computing. Because Java objects can be run from any platform, the idea behind RMI is to just write everything in Java and then have those objects communicate with each other.

Although Java can be used to write CORBA objects that can be called over IIOP, or even to write COM objects using certain nonstandard Java language extensions, using RMI for distributed computing can provide a smaller learning curve because the programmer isn't required to learn about CORBA and IIOP. All of the objects involved use the same programming language, so any data types are simply the built-in Java datatypes, and Java exceptions can be used for error handling. Finally, Java RMI can do one thing DCOM and IIOP can't: It can transfer code with every call. That is, even when the remote computer you're calling doesn't have the code it needs, you can send it and still have the remote computer perform the processing.

The obvious drawback to Java RMI is that it ties the programmer to one programming language, Java, for all of the objects in the distributed system.

The New Middleware Protocol: Web Services

With the Internet fast becoming the platform on which applications run, it's no surprise that a truly language- and platform-independent way of creating distributed applications has become the Holy Grail of software development. Currently, it looks as though that Holy Grail has made itself known in the form of web services.

Web services are a means for requesting information or carrying out a processing task over the Internet, but, as stated, they typically involve the encoding of both the request and the response in XML. Along with using standard Internet protocols for transport, this encoding makes messages universally available. That means that a Perl program running on Linux can call a .NET program running on Windows.NET, and nobody will be the wiser.

1.3. Web Services

Web services are open standard (XML, SOAP, HTTP, etc.) based web applications that interact with other web applications for the purpose of exchanging data

WHAT ARE WEB SERVICES?

Different books and different organizations provide different definitions to Web Services. Some of them are listed here.

- ✓ A web service is any piece of software that makes itself available over the internet and uses a standardized XML messaging system. XML is used to encode all communications to a web service. For example, a client invokes a web service by sending an XML message, then waits for a corresponding XML response. As all communication is in XML, web services are not tied to any one operating system or programming language—Java can talk with Perl; Windows applications can talk with Unix applications.

- ✓ Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML.

- ✓ Web services are XML-based information exchange systems that use the Internet for direct application-to-application interaction. These systems can include programs, objects, messages, or documents.

- ✓ 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.

To summarize, a complete web service is, therefore, any service that:

- Is available over the Internet or private (intranet) networks
- Uses a standardized XML messaging system
- Is not tied to any one operating system or programming language
- Is self-describing via a common XML grammar
- Is discoverable via a simple find mechanism