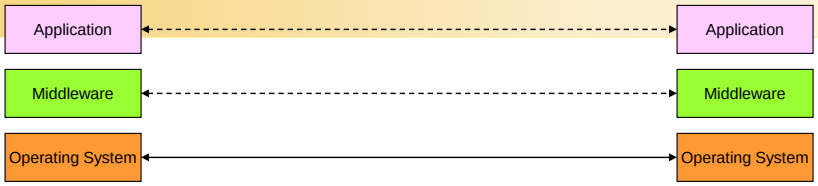
What is Middleware?

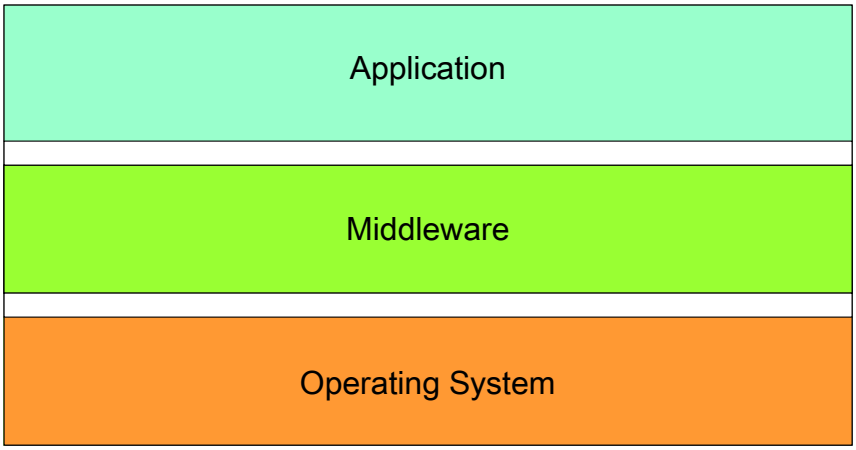


Software that functions as a conversion or translation layer.

- It is also a consolidator and integrator.
  - Custom-programmed middleware solutions have been developed for decades to enable one application to communicate with another that either runs on a different platform or comes from a different vendor or both.
- Today, there is a diverse group of products that offer packaged middleware solutions.

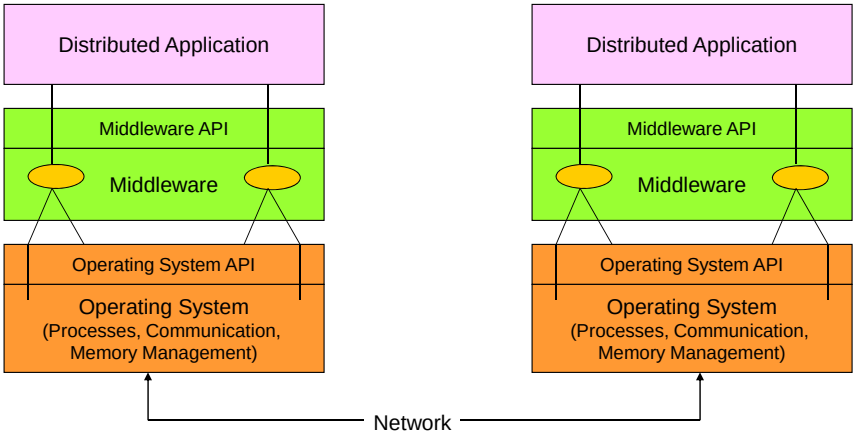
AP 04/07

Operating System vs. Middleware



AP 04/07

The Middleware Layer

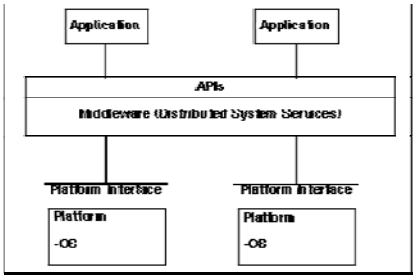


Purpose and Origin

- Middleware is connectivity software that consists of a set of enabling services that allow multiple processes running on one or more machines to interact across a network.
- Middleware is essential to migrating mainframe applications to client/server applications and to providing for communication across heterogeneous platforms.
  - This technology has evolved during the 1990s to provide for interoperability in support of the move to client/server architectures (see Client/Server Software Architectures).
- The most widely-publicized middleware initiatives are the
  - Open Software Foundation's Distributed Computing Environment (DCE),
  - Object Management Group's Common Object Request Broker Architecture (CORBA), and
  - Microsoft's COM/DCOM (Component Object Model)

Technical Detail

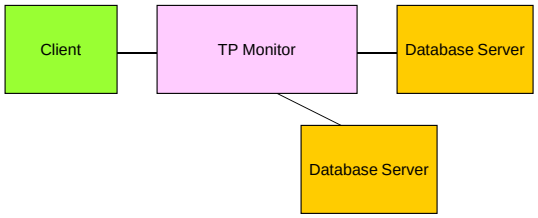
- Middleware services are sets of distributed software that exist between the application and the operating system and network services on a system node in the network.



Middleware Services

- provide a more functional set of Application Programming Interfaces (API) than the operating system and network services to allow an application to
  - locate transparently across the network, providing interaction with another application or service
  - be independent from network services
  - be reliable and available
  - scale up in capacity without losing function

TP Monitors
(Transaction Processing Monitors)



- First product to be called middleware.
- Sitting between the requesting client program and the databases,
  it ensures that all databases are updated properly.

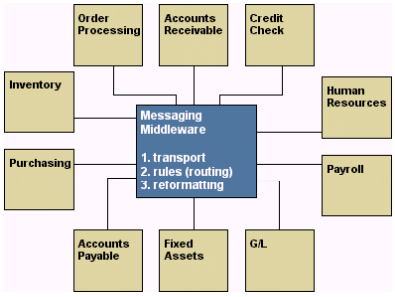
TP Monitors‘ Characteristics

- Tend to do far more than coordinate and monitor transactions across multiple data resources.
  - Enhance the performance, reliability, and scalability of server-side systems.
  - TP monitors establish a framework for creating server-side applications.
  - A TP monitor can reliably and efficiently manage the resources needed by applications that conform to the TP monitor’s rules.
- CICS (Customer Information Control System) and IMS/TM (a message-based transaction manager) are the transaction processing workhorses of the mainframe environment.
- On UNIX systems, BEA’s TUXEDO, BEA’s TOP END, and IBM’s Encina are the most widely used TP monitors.

Messaging Middleware

Common interface and transport between applications.

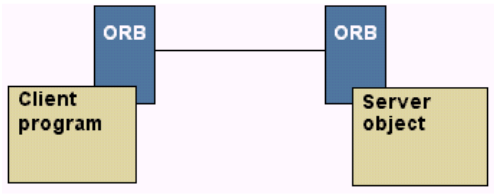
- Stores the data in a message queue if the target machine is down or overloaded
- May contain business logic that routes messages to the appropriate destinations and reformats the data as well.
- Similar to an e-mail messaging system, except that it is used to send data between applications.



Messaging Middleware Products

- The predominant messaging product for managing asynchronous communication between applications is **IBM's MQSeries**.
  - MQSeries has been ported to all major server platforms.
- In conjunction with the Component Object Model (COM), Microsoft introduced its own messaging system, **Microsoft Message Queue Server (MSMQ).**
  - MSMQ and MQSeries offer much the same functionality.

Distributed Processing

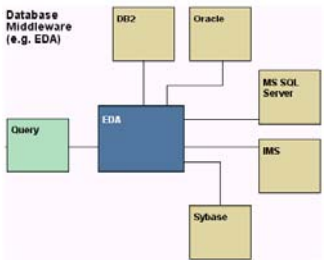


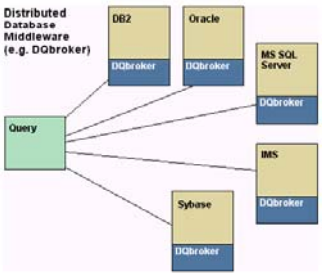
- Distributed object systems such as CORBA, DCOM and EJB enable processes to be run anywhere in the network.
- They differ from messaging middleware in that they cause processes (components/objects) to be executed in a synchronous fashion rather than sending data asynchronously.

Remote Procedure Calls

- *Remote Procedure Calls* (RPCs) enable the logic of an application to be distributed across the network.
- Most prominent examples:
  - SUN RPC, introduced with the network file system (SUN NFS),
  - DCE RPC, served as technical foundation of Microsoft's COM.
- *Object Request Brokers* (ORBs) enable the objects that comprise an object-oriented application to be distributed and shared across heterogeneous networks.
  - Extending the procedural programming model of RPC,
  - Distributed object systems such as CORBA, DCOM, .NET, and EJB enable processes to be run anywhere in the network.

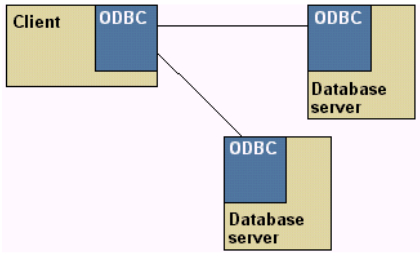
Database Middleware





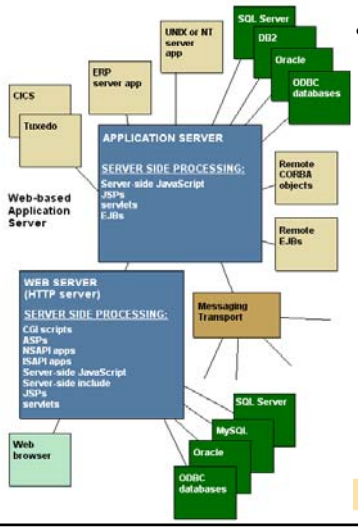
- Database Middleware provides a common interface between a query and multiple, distributed databases.
- Using either a hub and spoke architecture or a distributed architecture it enables data to be consolidated from a variety of disparate data sources

Common Interfaces



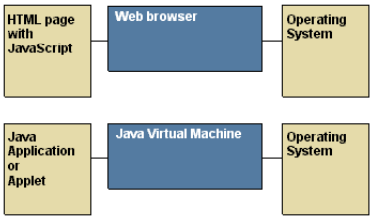
- Common programming interfaces between applications are considered middleware.
  - Open Database Connectivity (ODBC) enables applications to make a standard call to all the databases that support the ODBC interface.

Application Server Middleware



- A Web-based application server that provides interfaces to a wide variety of applications is used as middleware between the browser and legacy systems.
  - The browser can be used at desktops or on laptops when travelling.
  - A wide range of server-side processing has been supported by appservers (i.e.;J2EE).

Universal Computing



The Holy Grail of computing.

- Enable the same program to run on any hardware platform without modification.
- HTML pages written in JavaScript can execute on any JavaScript-enabled Web browser running under any operating system.
- Java applications and applets are executed by a Java Virtual Machine, which can be created for any operating system.

Browser and Java negate the requirement for a single operating system and hardware environment.

Usage Considerations

- The main purpose of middleware services is to help solve many application connectivity and interoperability problems. However, middleware services are not a panacea:
  - There is a gap between principles and practice. Many popular middleware services use proprietary implementations (making applications dependent on a single vendor's product).
  - The sheer number of middleware services is a barrier to using them. To keep their computing environment manageably simple, developers have to select a small number of services that meet their needs for functionality and platform coverage.
  - While middleware services raise the level of abstraction of programming distributed applications, they still leave the application developer with hard design choices. For example, the developer must still decide what functionality to put on the client and server sides of a distributed application.

Types of Middleware Services

1. Distributed system services,
   - Critical communications, program-to-program, and data management services.
   - This type of service includes RPCs, MOMs and ORBs.
2. Application enabling services,
   - Access to distributed services and the underlying network.
   - This type of services includes transaction processing monitors and database services such as Structured Query Language (SQL).
3. Middleware management services,
   - Which enable applications and system functions to be continuously monitored to ensure optimum performance of the distributed environment.

Distributed Objects and Distributed Processing

- Distributed objects have the biggest potential to solve a wide range of challenges faced by designers of large software systems.
- Some of these challenges include
  - component packaging,
  - cross-language interoperability,
  - interprocess communication, and
  - intermachine communication.
- We separate distributed object architectures into two categories:
  - component architectures and
  - remoting architectures.
- Component architectures focus primarily on component packaging and cross-language interoperability.
- Remoting architectures focus primarily on support for remote method invocation on distributed objects.

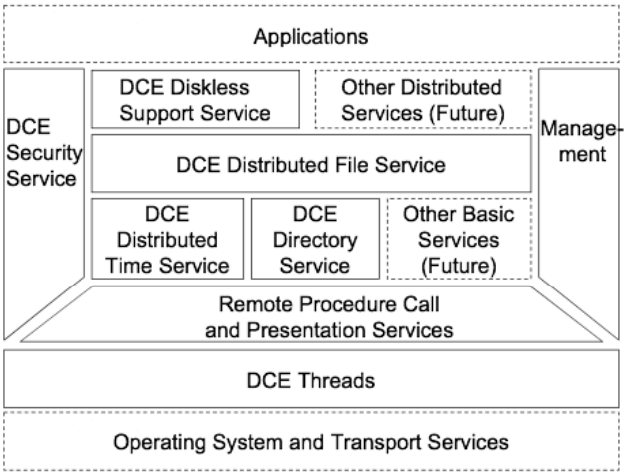
Remoting Architectures

- Open Software Foundation's (OSF) Distributed Computing Environment (DCE)
  - which actually is a distributed processing environment based on the Remote Procedure Call (RPC) paradigm (purely procedural)
- Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA).
  - The notion of component packaging and deployment has only recently been added to CORBA 3.0.

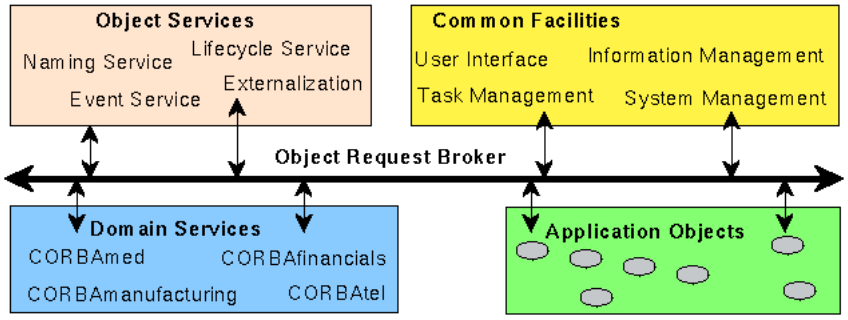
Component Architectures

- Microsoft's Component Object Model (COM)
  - addresses packaging and deployment of binary component as well as cross-language interoperability
- JavaBeans and Enterprise Java Beans (EJB) component models introduced by SUN Microsystems.
- Both, COM and EJB address remoting to some extend:
  - the COM model has been extended to Distributed COM (DCOM) using an extended version of DCE RPC as transport.
  - EJB supports client/server communication based on Java Remote Method Invocation (RMI).
  - RMI is special as it integrates closely with the Java language without requiring a special Interface Definition Language (IDL) to describe component interfaces accessible for remote invocations.
- In an evolutionary sense, Microsoft's .NET is the newest and most advanced component architecture available in the market today.

DCE Architecture and Services

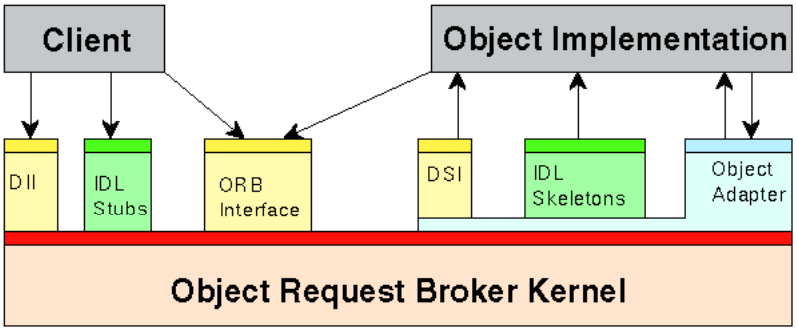


The Object Management Architecture



AP 04/07

ORB Interfaces



AP 04/07

Recent CORBA 3.0 Additions

- **CORBA Component Model (CCM)** –
  - introduces the notion of server-side components into CORBA and addresses packaging and deployment for CORBA components.
  - CCM provides for interoperability with EJB.
- **Objects passable by value (valuetypes)** –
  - valuetypes definitely improve integration with Java and also serve as basis for the XML/Value mapping (released as part of CORBA 2.3).
- **Java-to-IDL Mapping** –
  - this mapping allows Java RMI objects to interoperate over the network like CORBA objects using CORBA object references.
- **XML/Value mapping** –
  - standardizes the representation of an XML document as a collection of native CORBA types.
- **CORBA Firewall Specification** –
  - allows firewalls to be configured for CORBA using access rules for IIOP traffic.

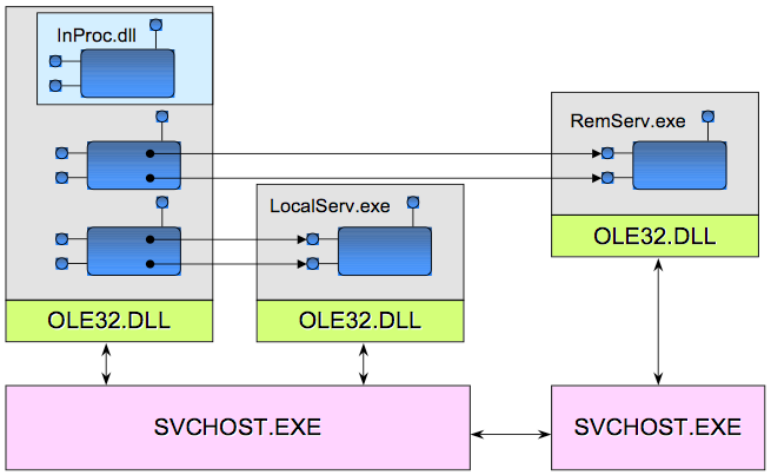
AP 04/07

CORBA 3.0 Additions

- **CORBA Messaging** –
  - encompasses both asynchronous and messaging-mode invocations of CORBA objects as well as Quality of Service Control.
- **Real-Time CORBA** –
  - extends the CORBA architecture with resource control mechanisms for real-time applications running on a real-time operating system in a controlled environment.
- **Fault-Tolerant CORBA** –
  - standardizes redundant software configurations and systems that give CORBA robust and reliable performance (when run on redundant hardware).
- **Minimum CORBA** –
  - defines a small-footprint CORBA configuration that is aimed at embedded and card-based systems.

AP 04/07

Styles of COM Server Components



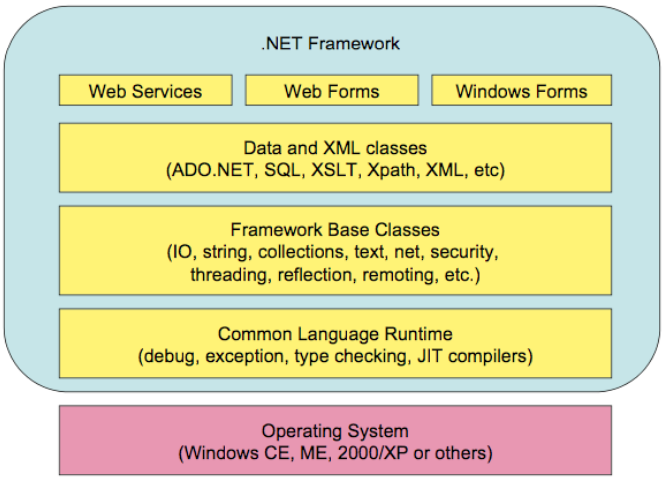
AP 04/07

COM Characteristics

- **COM** is a very mature component architecture that has many strengths.
  - Thousands of third-party ActiveX controls (in-process COM components) are available in the market today.
  - Microsoft and other vendors have built many tools that accelerate development of COM-based applications.
- Advanced services such as **Microsoft Transaction Server (MTS)** and **Microsoft Message Queuing Server (MSMQ)** support development of enterprise multi-tier systems.
  - Microsoft has been using the name COM+ to identify the bundling of the COM runtime with those services.

AP 04/07

Microsoft .NET Framework



AP 04/07

.NET Characteristics

- **Distributed Computing**:
  - .NET provides a remoting architecture that exploits open Internet standards, including the Hypertext Transfer Protocol (http), Extensible Markup Language (XML), and Simple Object Access Protocol (SOAP).
- **Componentization**:
  - .NET extends the previous COM component model but provides a significantly simpler way to build and deploy components.
- **Enterprise services**:
  - .NET supports the development of scalable enterprise applications without writing code to manage transactions, security, or pooling.
- **Web paradigm shifts**:
  - Over the last few years, web application development has shifted from connectivity (TCP/IP), to presentation (HTML), to programmability (XML and SOAP). .NET enables software to be sold and distributed as a service.
- **Maturity factors**:
  - Although .NET is a relatively new framework, it builds upon the mature COM+ technology and services.

AP 04/07

Future Trends: Resource Management and Quality-of-Service

- Middleware abstractions provide resource management in a distributed system at a high level.
  - OS manages: communication, processing, storage (memory/disks).
  - Middleware abstractions also are from an end-to-end perspective, not just of a single host, which allows for a more global and complete view to a resource management system.
  - Distributed objects are promising, as they not only encapsulate but also cleanly integrate all three kinds of resource into a coherent package.
  - This completeness helps distributed resource management and makes it easier to provide for load balancing, mobility transparency, and overall system reliability.

New Capabilities

- Recent middleware frameworks, like the CORBA 3.0 Component Model (CCM), or the Microsoft .NET framework, allow the expression of non-functional component properties.
  - such as resource requirements, timing and security constraints, or fault-tolerance assumptions on the component level using language constructs (like C# and .NET attributes) or component meta-data (like CCM's deployment descriptors).
- Aspect-Oriented Programming (AOP) is a relatively new discipline that focuses on cross-cutting concerns
  - targeting many components of a system simultaneously and
  - non-functional component properties.
- AOP investigates software engineering approaches towards predictable component-based systems.
  - This research opens up new venues for middleware-based architectures on the enterprise level.

Middleware hides Complexity

- “Any problem in Computer Science can be solved with another level of indirection”
  - Butler Lampson
- “Except the problem of indirection complexity”
  - Bob Morgan

References and Information Sources

- [Bernstein 96] Bernstein, Philip A. "Middleware: A Model for Distributed Services." Communications of the ACM 39, 2 (February 1996): 86-97.
- [Client 95] "Middleware Can Mask the Complexity of your Distributed Environment." Client/Server Economics Letter 2, 6 (June 1995): 1-5.
- [Eckerson 95] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." Open Information Systems 10, 1 (January 1995): 3(20).
- [Schreiber 95] Schreiber, Richard. "Middleware Demystified." Datamation 41, 6 (April 1, 1995): 41-45.