Chapter 11

Distributed Systems Architectures

Architectural design for software that executes on more than one processor

Topics covered

- Multiprocessor architectures
- Client-server architectures
- Distributed object architectures
- CORBA

Distributed systems

- Virtually all large computer-based systems are now distributed systems.
- Information processing is distributed over several computers rather than confined to a single one.
- Distributed software engineering is now very important.

System types

- Personal systems are those designed to run on a single personal computer or workstation.
- Distributed systems are those designed to run on a network of cooperating processors.
- Embedded systems are those embedded in a larger hardware system, and must respond in real time to events occurred in the environment.

Distributed system characteristics

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault tolerance
- Transparency

Distributed system disadvantages

- Complexity
- Security
- Manageability
- Unpredictability

Design issues:
resource identification

- Resources are available on different computers in different locations
- An appropriate naming scheme has to be devised so that each item can be named and referenced (such as Universal Resource Locator or URL on WWW)
- The naming scheme has to be meaningful and well understood

Design issues:
Communications

- For most applications, the Internet is the most efficient way for communications among the components.
- There might be cases in which alternative means for system interconnection are more appropriate.

Design issues:
quality of service

- The quality of service will be affected by many factors, among them:
  - allocation of processes to processors
  - distribution of resources
  - selection of network and system hardware

Design issues:
software architecture

- The way the application functionality is distributed over a number of logical components, and the way these components are allocated to the processors define the software architecture
- Right architecture is needed to achieve the desired quality of service.

Distributed systems architectures

- Client-server architectures

  Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services.
- Distributed object architectures

  No distinction between clients and servers. Any object on the system may provide and use services from other objects.

The need for middleware

- The different components in a distributed system may be implemented in different languages and may execute on completely different type of processors.
- Data or information representation as communication protocol used in the network may all be different.
- Software designed to make these diverse parts to work together is called *middleware*.

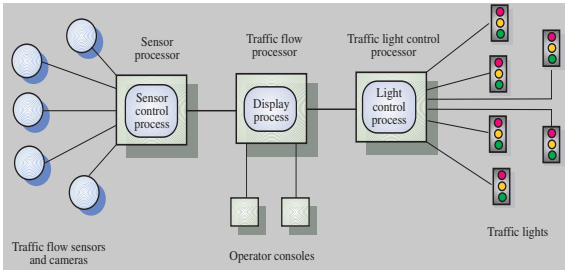
Middleware

- Software that manages and supports various components in a distributed system. In essence, it sits in the *middle* of the system.
- Middleware is usually off-the-shelf rather than specially written software.
- Examples:
  - Transaction processing monitors
  - Data converters
  - Communication controllers

Multiprocessor architectures

- Simplest distributed system model
- Composed of multiple processes which may execute on different processors
- Architectural model of many large real-time systems
- Distribution of processes to processors may be pre-ordered or may be under the control of a dispatcher

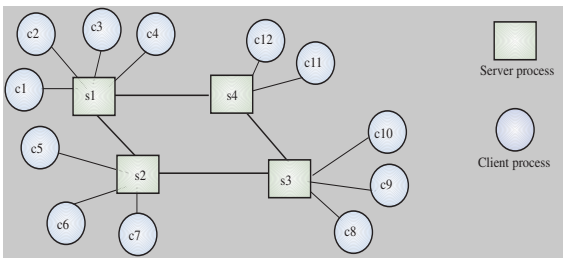
A multiprocessor traffic control system



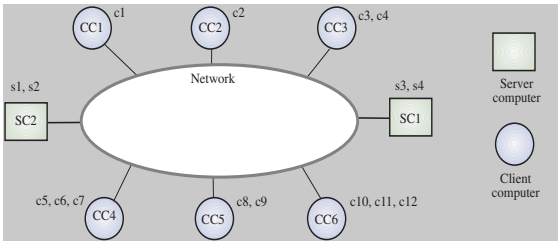
Client-server architectures

- An application is modelled as a set of services that are provided by servers and a set of clients that use these services.
- Clients know of servers but servers need not know of clients.
- Clients and servers are concurrent processes.
- More than one process may reside in a processor.

A client-server system



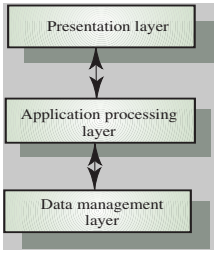
Computers in a client-server network



Layered application architecture

- Presentation layer
  Concerned with presenting the results of a computation to system users and with collecting user inputs
- Application processing layer
  Concerned with providing application specific functionality e.g., in a banking system, banking functions such as open account, close account, etc.
- Data management layer
  Concerned with managing the system databases

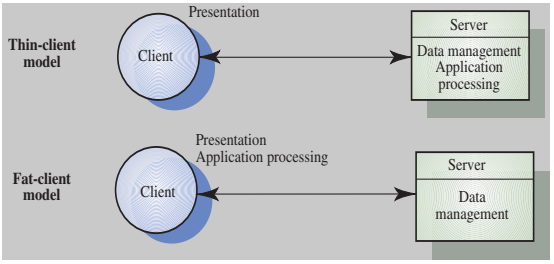
Application layers



Thin and fat clients

- *Thin-client model*
  All of the application processing and data management is carried out on the server. The client is simply responsible for running the presentation software.
- *Fat-client model*
  The server is only responsible for data management. The software on the client implements the application processes and user interface.

Thin and fat clients



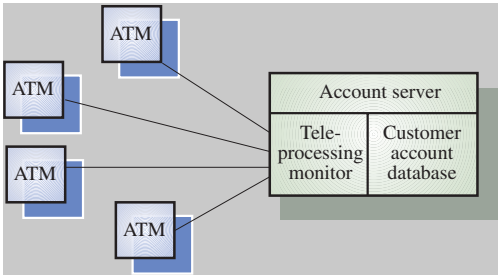
Thin client model

- Used when legacy systems are migrated to client server architectures.
  The legacy system acts as a server in its own right with a graphical interface implemented on a client.
- A major disadvantage is that it places a heavy processing load on both the server and the network.

Fat client model

- More processing is delegated to the client as the application processing is locally executed.
- Most suitable for new client-server systems where the capabilities of the client system are known in advance.
- More complex than a thin client model especially for management. New versions of the application have to be installed on all clients.

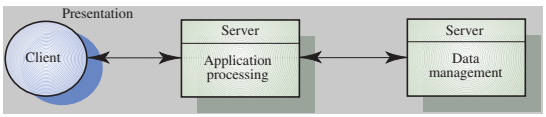
A client-server ATM system



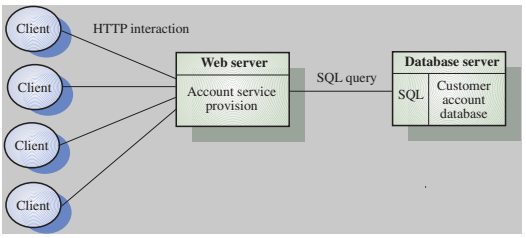
Three-tier architectures

- In a three-tier architecture, each of the application architecture layers may execute on a separate processor.
- Allows for better performance than a thin-client approach and is simpler to manage than a fat-client approach.
- A more scalable architecture - as demands increase, extra servers can be added.

A 3-tier client-server architecture



An internet banking system



Two-tier C/S with thin clients

- Legacy system applications in which processing and data management functions cannot be easily separated
- Computation-intensive applications that requires little or no data management activities, such as compilers
- Data intensive applications with little or no processing, such as browsing or querying by library users

Two-tier C/S with fat clients

- Applications with COTS (such as MS Excel) on the clients
- Applications with intensive data processing (such as data visualization)
- Applications with stable functionality and well-established system management (such as programmers workstations)

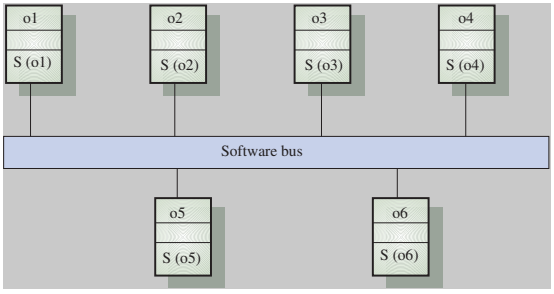
Multi-Tier Client-Server architectures

- Large-scale applications with hundreds or thousands of clients
- Applications where both the data and application are volatile
- Applications where data from multiple sources are integrated

Distributed object architectures

- In a distributed object architecture, there is no distinction between clients and servers.
- Each distributable entity is an object that provides services to other objects and receives services from other objects.
- Object communication is through a middleware system called an object request broker (software bus)
- More complex to design than client/server systems.

Distributed object architecture



Advantages of distributed object architecture

- It allows the system designer to delay decisions on where and how services should be provided.
- It is a very open system architecture that allows new resources to be added to it as required.
- The system is flexible and scaleable.
- It is possible to reconfigure the system dynamically with objects migrating across the network as required.

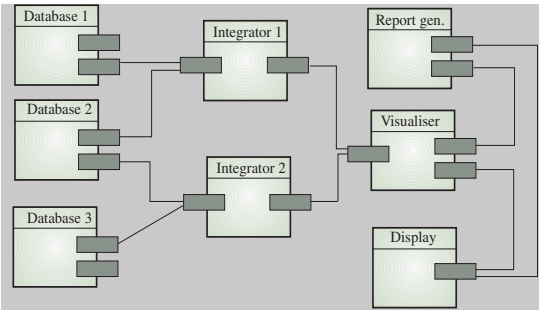
Uses of distributed object architecture

- As a logical model that allows you to structure and organize the system. In this case, you think about how to provide application functionality solely in terms of services and combinations of services.
- As a flexible approach to the implementation of client-server systems. The logical model of the system is a client-server model but both clients and servers are realized as distributed objects communicating through a software bus.

Example: a data mining system

- The logical model of the system is not one of service provision where there are distinguished data management services.
- It allows the number of databases that are accessed to be increased without disrupting the system.
- It allows new types of relationship to be mined by adding new integrator objects.

Depicting a data mining system



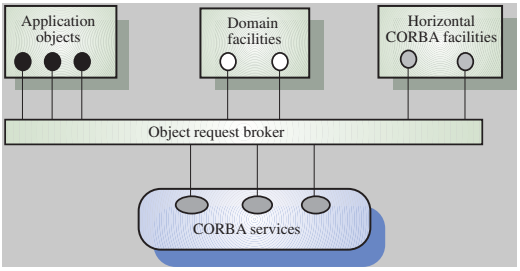
CORBA

- CORBA is an international standard for an Object Request Broker - middleware to manage communications among distributed objects.
- Several implementation of CORBA are available.
- CORBA has been defined by the Object Management Group.
- DCOM is an alternative approach to object request brokers developed by Microsoft.

Application structure

- Application objects
- Standard objects, defined by the OMG for a specific domain, e.g., insurance
- Fundamental CORBA services such as directories and security management
- Horizontal (i.e. cutting across applications) facilities such as user interface facilities

CORBA application structure



CORBA standards

- An object model for application objects
  A CORBA object is an encapsulation of state with a well-defined, language-neutral interface defined in an IDL (interface definition language)
- An object request broker that manages requests for object services.
- A set of general object services of use to many distributed applications.
- A set of common components built on top of these services.

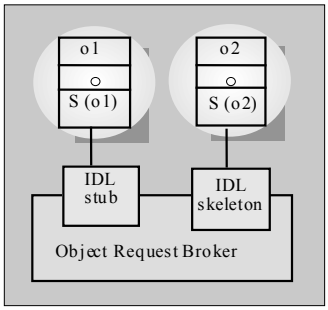
CORBA objects

- CORBA objects are comparable, in principle, to objects in C++ and Java.
- They MUST have a separate interface definition that is expressed using a common language (IDL) similar to C++.
- There is a mapping from this IDL to programming languages (C++, Java, etc.).
- Objects written in different languages, therefore, can communicate with each other.

Object request broker (ORB)

- The ORB handles object communications. It knows of all objects in the system and their interfaces.
- Using an ORB, the calling object binds an IDL stub that defines the interface of the called object.
- Calling this stub results in calls to the ORB which then calls the required object through a published IDL skeleton that links the interface to the service implementation.

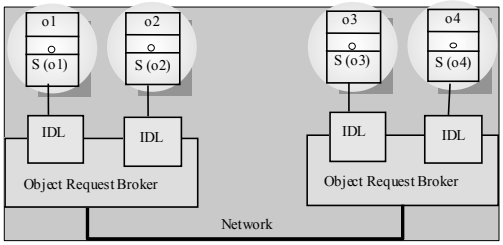
ORB-based object communications



Inter-ORB communications

- ORBs are not usually separate programs but are a set of objects in a library that are linked with an application when it is developed.
- ORBs handle communications between objects executing on the same machine.
- Several ORBS may be available and each computer in a distributed system will have its own ORB.
- Inter-ORB communications are used for distributed object calls.

Inter-ORB communications



CORBA services

- Naming and trading services

  These allow objects to discover and refer to other objects on the network.
- Notification services

  These allow objects to notify other objects that an event has occurred.
- Transaction services

  These support atomic transactions and rollback on failure.

Key points

- Almost all new large systems are distributed systems.
- Distributed systems support resource sharing, openness, concurrency, scalability, fault tolerance and transparency.
- Client-server architectures involve services being delivered by servers to programs operating on clients.
- User interface software always runs on the client and data management on the server.

Key points (continued)

- In a distributed object architecture, there is no distinction between clients and servers.
- Distributed object systems require middleware to handle object communications.
- The CORBA standards are a set of middleware standards that support distributed object architectures.