The UNIVERSITY *of* EDINBURGH
SCHOOL *of* COMPUTER SCIENCE

CS4/MSc

Distributed Systems

Jane Hillston

jeh@dcs.ed.ac.uk

Room 1604

(Lecture 1: Introduction, 7th January 2002)

What is a distributed system?

A distributed system is one in which I can be prevented from getting on with my work because a computer I have never heard of has crashed....

A distributed system is broadly categorised as a collection or network of loosely coupled, autonomous computers with the following characteristics:

- The nodes are relatively loosely coupled.
- Each node is a self-contained autonomous computer with its own peripherals.
- The system can survive various categories of node and network failures.
- The nodes may execute logically separate computations, though these may be related to concurrent computations on other nodes.
- The system is asynchronous.

Overview of the course

Distributed Computation: characteristics of distributed systems, overview of related concepts (networks, operating systems, etc.)

Interprocess Communication: message passing, client/server communication model, remote procedure call (RPC)

Distributed Objects and Remote Invocation: distributed object technologies such as CORBA, Java RMI and DCOM.

Distributed File Systems and Name Services: file services, name services and directory and discovery services

Time and Global States: physical/logical clocks, synchronisation

Coordination and Agreement: distributed agreement in the presence of failures, mutual exclusion, leader election

Distributed Transactions: timestamp ordering, atomic commit protocols and distributed deadlocks

Emerging Technologies: mobile code and mobile agents, peer-to-peer systems and ubiquitous networking.

Recurring Themes

Concurrency One of the major motivations of distributed systems is resource sharing. When this sharing takes place between concurrently executing programs coordination is needed. Efficiency can be improved by adding more resources.

No global clock Since all communication is via message passing absolute synchronisation of clocks is not possible. This can create difficulties for cooperation between processes.

Independent failures Problems arise not just because failures can occur in any of the components or the network of a distributed system, but also failures may not be readily detected.

Basic notions

RESOURCE *something that can be usefully shared in a networked computer system* e.g. data, hardware, application,...

SERVICE part of a computer system which manages a collection of related resources and presents their functionality to users and applications; access is restricted to a well-defined set of *operations*.

SERVER a program or process on a networked computer which provides a service.

CLIENT a program or process generating a request to a server (*invocation*); requests are sent in messages from the client to the server and replies are sent in messages from the server to the client.

Note that *client* and *server* apply only to the roles played within a single invocation.

PEER PROCESSES processes that cooperate and communicate in a symmetrical manner to perform a task.

Challenges

A number of challenges are recognised in the design of distributed systems, and these have been tackled with varying degrees of success in existing systems. In some cases they place conflicting requirements on the system. The list below gives some indication of measures that are employed to meet each challenge:

- Heterogeneity—*standards and protocols; middleware; virtual machine;*
- Openness—*publication of services; notification of interfaces; independence from individual vendors;*
- Security—*firewalls; encryption*
- Scalability—*replication; caching; multiple servers;*
- Failure Handling—*failure tolerance; recover/roll-back; redundancy;*
- Concurrency—*concurrency control to ensure data consistency;*
- Transparency—*middleware; location transparent naming; anonymity*

ISO Reference Model for Open Distributed Processing

Aims to conceal the component-based structure of the system, and facilitate a perception of the system as a whole:

- Access Transparency
- Location Transparency
- Concurrency Transparency
- Replication Transparency
- Failure Transparency
- Mobility/Migration Transparency
- Performance Transparency
- Scaling Transparency

Access and location transparency together are considered to provide *network transparency*.

Architectural Models

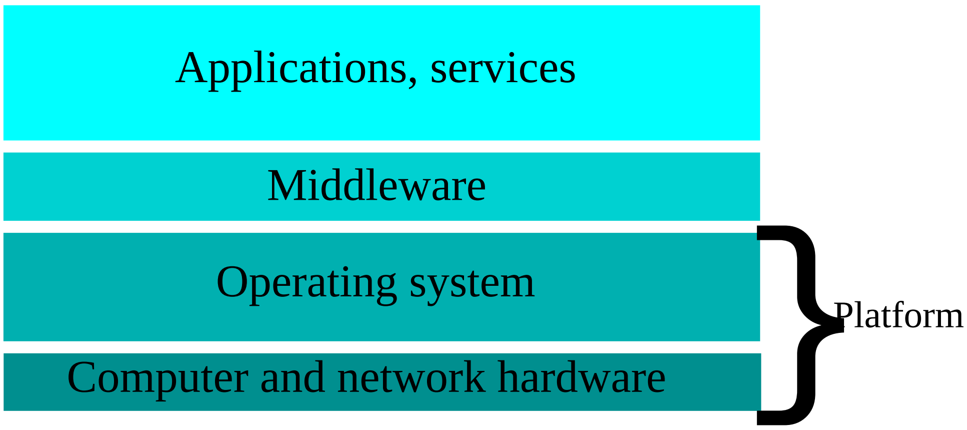
Architectural models are used to show the possible ways in which components of a distributed system may interact. They are useful for highlightling:

- the placement of components across a network of computers, providing useful patterns for the distribution of data and workload.
- the inter-relationships between the components, outlining their functional roles and the patterns of communication between them.

In the following slides we examine several such models.

Service Layers in Distributed Systems

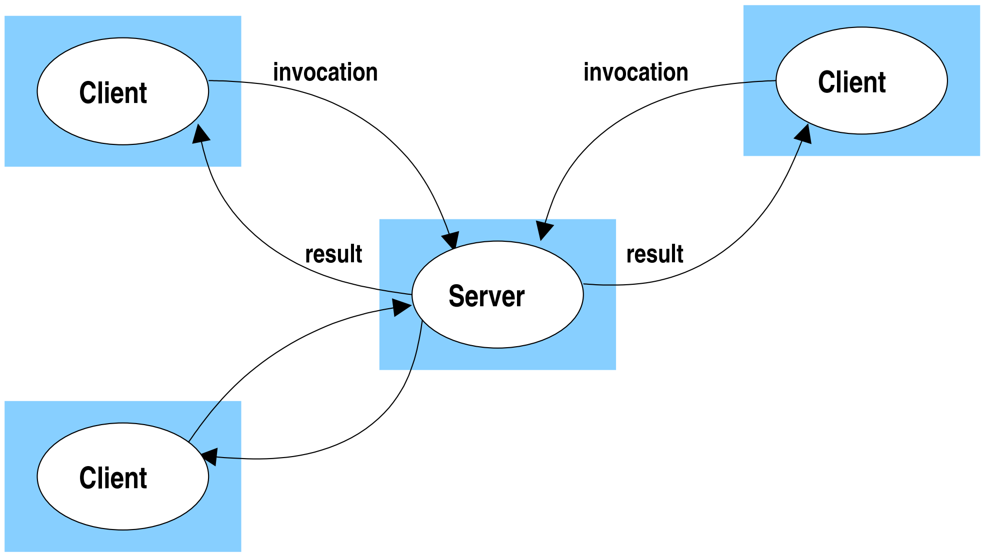
In the layered view of a system each layer offers services to the level above and builds its own service on the services of the layer below.



The operating system and the hardware together are often referred to as the *platform*. The *middleware* layer is intended to hide heterogeneity and provide a useful abstraction for application programmers.

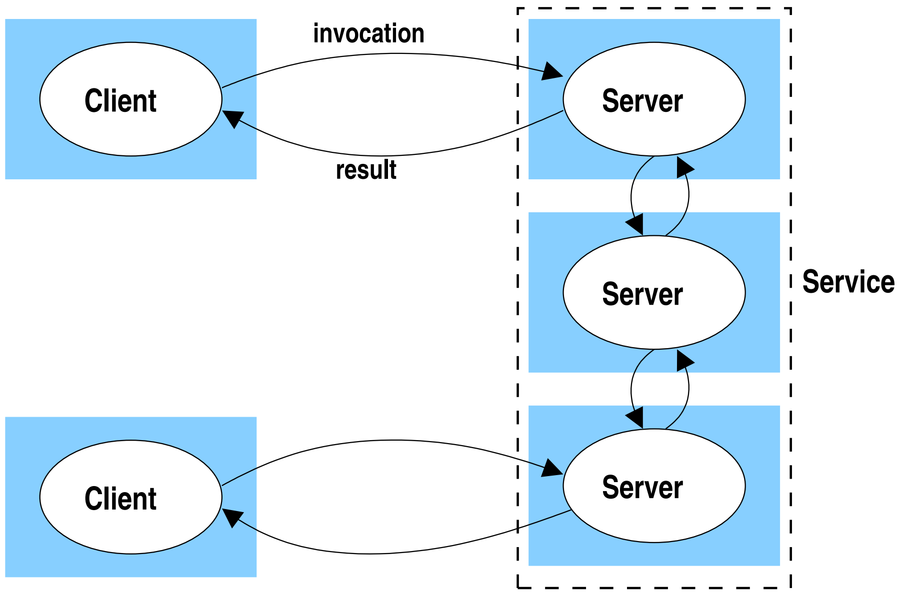
Client-Server Model (Simple Case)

In the simplest case clients invoke single servers which satisfy their requests individually:



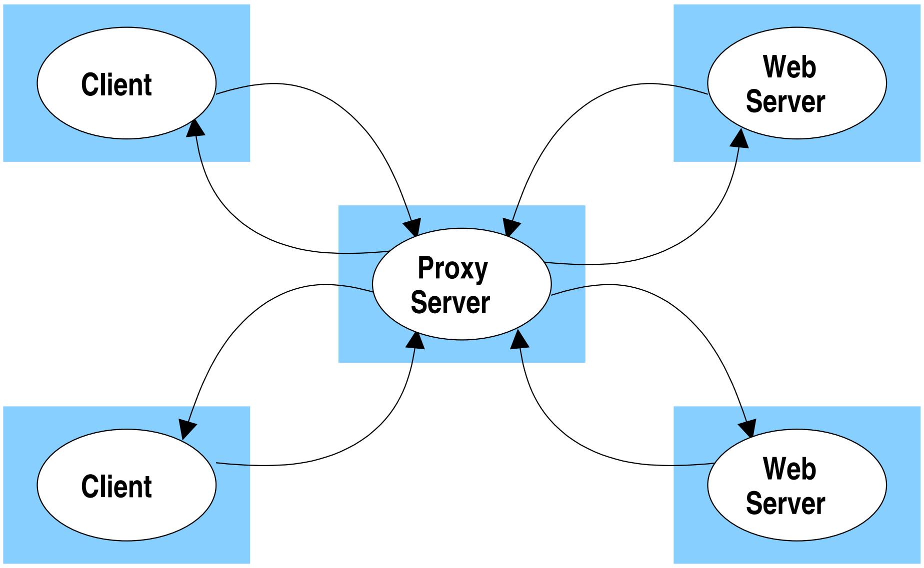
Alternative Client-Server Model

A more general case allows service to be provided by multiple servers: servers on distinct hosts interact as appropriate to provide service.



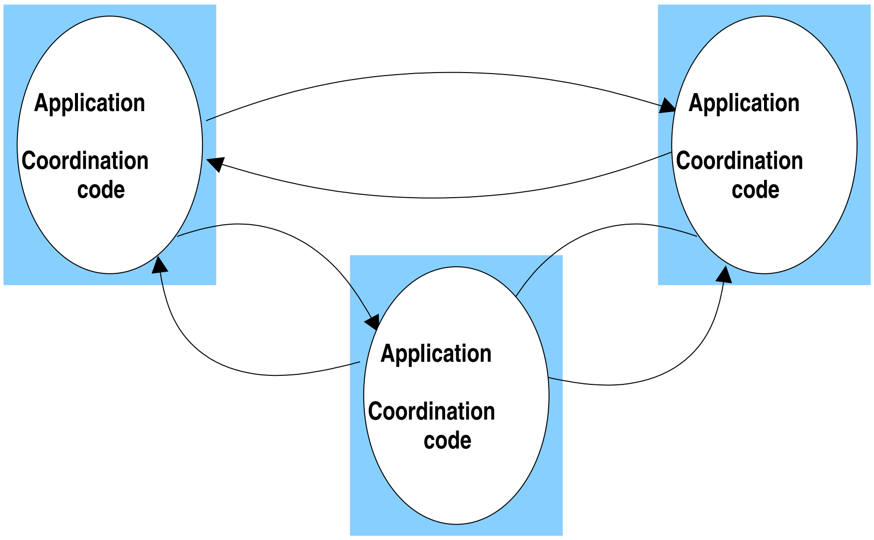
Proxy Servers and Caches

A *cache* is used to avoid performance bottlenecks and reduce network traffic: data resources are moved closer to the clients. A proxy server may host a cache.



Peer Processes

All processes play similar roles and result is achieved by cooperation.

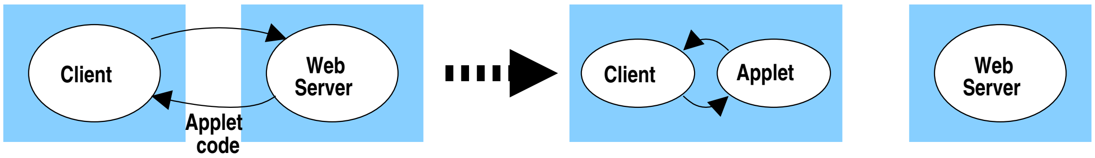


Pattern of communication will depend on the application requirements.

Impact of Mobility on Client-Server Model

Mobility, both of code and devices, has altered the way that people think about client-server systems. There have been two major impacts:

- When mobile code is used, the code may be moved from one process to another, in effect allowing one process (e.g. a server) to delegate a task to another process (e.g. the client). For example, an applet is a piece of code which a client downloads from the server and runs locally. This can be used to reduce access delays and minimize communication costs.



Impact of Mobility (2)

- Mobility of devices mean that the configuration of a distributed system may be dynamic during the running of the system. Such systems are now being designed to allow computers and other mobile devices to be seamlessly added and removed (*spontaneous networking*). In these cases they need some means to *discover* the available services and *register* their own services.