DISTRIBUTED SYSTEMS

FUNDAMENTAL PROPERTIES

1. concurrent execution of components

2. independent failure modes
   components and connections may fail independently

3. transmission delay

4. relativistic time

IMPLICATIONS

cannot know why there's no reply - failure or congestion

cannot use local timestamps to order the occurrence of events

inconsistent state
  replicas of a single object
  related changes to different objects

DISTRIBUTED SOFTWARE SYSTEMS

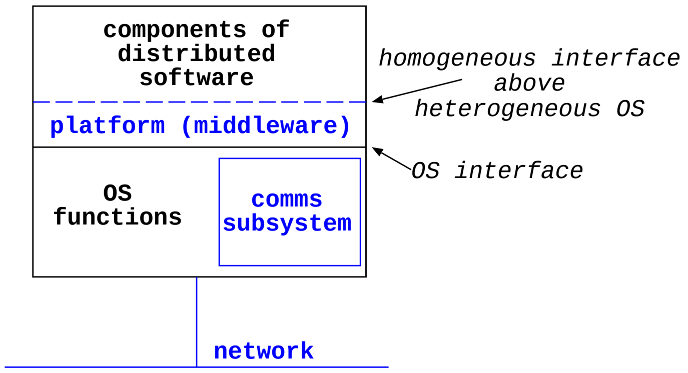
* what is a "conventional" system?

* previous courses have covered general principles of operating systems and communications

* we now look explicitly at support for distributed software which may be:

  - directly by OS in a homogeneous cluster

  - by a platform (part of middleware) (ORB if O-O) above heterogeneous OS.



DISTRIBUTED SYSTEMS POTENTIAL

(from about 1985)

* cost effectiveness

many small machines cf. few large ones

* high performance

real concurrency cf. timesharing - better response
distributed algorithms to exploit concurrency

* availability

if one processor fails the others can
continue .... MAYBE ....

* extensibility

- incremental growth rather than replacement
- design of a range of configurations

* simplicity due to physical modularity

- may have dedicated processors in some
  applications e.g. process control

DISTRIBUTED SYSTEMS, EXAMPLES

* process control

manufacturing,

* automation of commerce

stock control,invoicing, orders, sales

* utilities

billing, customer interaction,
fault reporting and repair, GIS

* public bureaucracies

health service, tax, social services

* financial systems

global transaction processing

* office systems

document storage, provision of applications,
groupware,.....

* distributed database systems

components of many systems,
e.g. telco's customer and 800 info
e.g. business systems such as SAP

* mobile users and devices

major growth area, challenge of huge scale

SPECIFIC DISTRIBUTED APPLICATIONS

e.g. email, news, ftp, - WWW?

client's email interface

names
for clients and messages
jmb@cl.cam.ac.uk
message-id

SMTP
simple mail
transfer protocol

specific,
application level
protocol

OS
comms interface

standard comms
supporting all applications

client's WWW interface

names are URLs
universal resource locators
http://www.cl.cam.ac.uk/....

HTTP
hypertext
transfer protocol

specific,
application level
protocol

OS
comms interface

* a browser interface may be used for distributed applications

+ simple, familiar and openly available

- problem of link management

GENERIC SUPPORT for DISTRIBUTED APPLICATIONS
(early example)

component of distributed application

names
interfaces - procedures

RPC
remote
procedure call

a general,
application level
protocol

OS
comms interface

standard comms
supporting all applications

RPC is an example of a general, application-level protocol above which distributed applications may be developed.

Middleware includes communications support which may be RPC and/or message passing.

Examples are ISO-ODP, OSF-DCE, OMG-CORBA, DCOM, Java-RMI

As we are beginning to see from the above, NAMES must be defined system-wide and middleware must support name-to-location mapping.

Middleware includes a communications support platform and services above the platform.

OPEN & PROPRIETARY SOFTWARE

* Open software

evolution is controlled by standards
bodies such as ISO or by consortia
such as OMG (Object Management Group).

RFPs (requests for proposals) are issued and
draft specifications are published with
RFCs (requests for comments)

a "standard" may be published without
implementations

* proprietary software

can be changed at the whim of the owner

consistency of functionality across versions
is not guaranteed

RELATED ISSUES

* single-language/multi-language?

can components be written in different
languages and interoperate?

* open interoperability

- is desirable across middleware platforms
  including different implementations of the same platform
  as well as between different programming languages

CLUSTER COMPUTING

* small-scale, homogeneous, single-language

may be used to support parallel algorithms (e.g. nuclear physics)

a possible architecture for "compute servers"

focus on resource-sharing, load balancing, accounting

may be based on a distributed shared-memory (DSM) abstraction

ASPECTS OF DISTRIBUTED SYSTEM DESIGN

MODEL of distributed computation

what are the named entities?
interfaces/procedures, objects, components,
structured documents
how is communication achieved?
synchronous invocation? asynchronous notification?

ARCHITECTURE - issues

naming
location of named objects
security of communication
authentication (recognition) of participants
protection (access control)
replication for reliability
heterogeneous/homogeneous - open interoperability

placement of functionality, coarse-grained
(in platform e.g. ORB or in supported entities?
client-server, 3-tier, peer-to-peer)

ENGINEERING issues

transparency of distribution
- location transparency
- failure transparency
- migration transparency
(transparency = HIDDEN -
can't be detected when running
needn't be programmed for explicity)
(may not be achievable in practice)
(which system level should provide
which transparency?)

placement of functionality, fine-grained
(libraries? wrappers?)

use of replication - achieving consistency

DISTRIBUTED SYSTEMS ARCHITECTURAL EVOLUTION

PROFESSIONAL COMPUTING

Distribution of function
between clients and servers:

terminals + multiaccess systems

terminals + pool of processors + servers
(e.g. Cambridge)

diskless workstations + servers
(e.g. Stanford)

workstations + servers
(e.g. Xerox PARC)

cluster-computing

large-scale, open interoperability

mobile users and computers

PERSONAL & UBIQUITOUS COMPUTING

standalone PCs

networked PCs

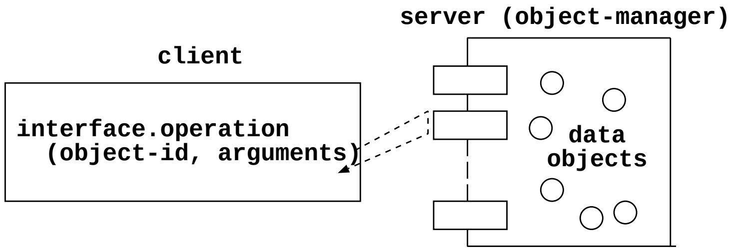
network computer
diskless + storage servers + processor-pool

thin clients + servers (Citrix)

laptops, palmtops, mobile devices,
network-enabled phones, active environments

Client-Server model

naming: of server interfaces and their operations



* everyone starts with the ability to invoke the name service (may be called interface trader)

* servers export their interfaces to the name service which supports name-to-location mapping for clients

* clients import interfaces from the name service

* clients invoke operations at servers

---

* a service may be provided by a number of servers

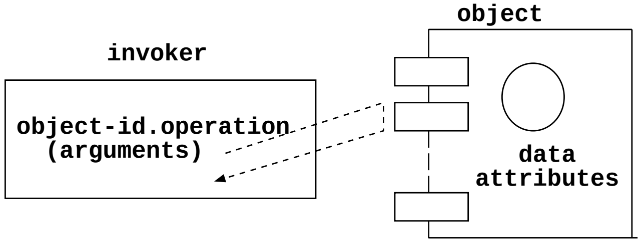
* a server may be an object manager
  - the objects may be persistent

* a server may be a client of other servers

e.g. client -> TPS -> File-service

Object model

NAMING: objects are named application/system-wide



* externally accessible typed objects may be registered with the middleware platform (ORB) which returns a globally valid id
* object id's may be used as first-class values
* on invocation, the platform (ORB) provides name-to-location mapping

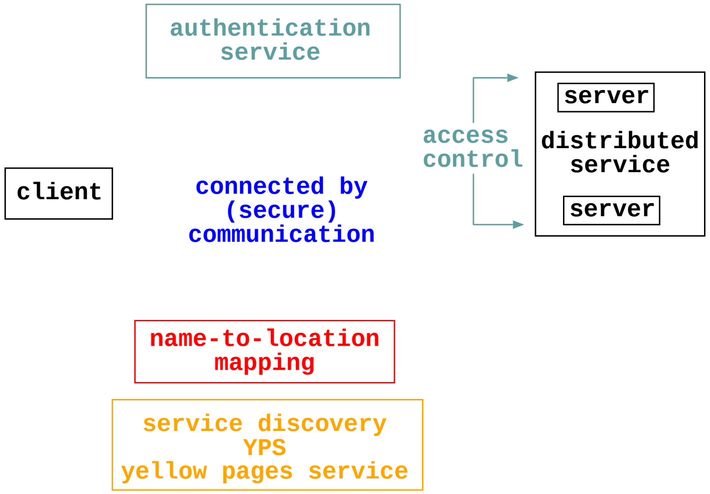
----

* the platform may support only remote invocation e.g. OMG CORBA
* the system may map objects into the invoker's address-space if a DSM (distributed shared memory) model is assumed.

  Appropriate only for homogeneous clusters of computers/OSs and a single-language system.

  e.g. Eden, Emerald, Somiw-SOS, Cool, Clouds, Comandos

DISTRIBUTED SYSTEMS ARCHITECTURAL COMPONENTS



authentication - client to server
server to client

access control - applied at server

naming - defined by system model

name-to-location binding
may be provided at service or platform level