

Distributed Systems

Ian Johnson

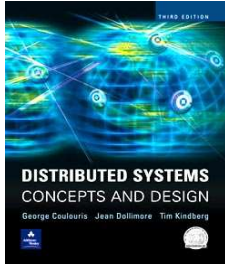
Room 3P16

Telephone: extension 3167

Email: Ian.Johnson@uwe.ac.uk

**[http://www.cems.uwe.ac.uk/
~irjohnso](http://www.cems.uwe.ac.uk/~irjohnso)**

Module is based on:



From **Coulouris, Dollimore and Kindberg**
Distributed Systems:
Concepts and Design
Edition 3, © Addison-Wesley 2001

WWW.CDK3.NET

Many of the diagrams etc. in my slides are taken from the above. It is strongly recommended you buy this!

Other useful sources:

- Beej's socket programming guide
(free pdf – sockets)
- O'Gorman, J; "*Operating Systems with linux*" Palgrave 2001.
(final 3 chapters concurrency, dist. file systems, fault tolerance & security)
- Orfali R& Harkey D; "*Client-Server programming with Java & Corba*", Wiley 1997.
- Richard, G.G.; "*Service & Device Discovery Protocols and Programming*", McGraw-Hill 2002.
- Edwards, W.K. "*Core JINI*"; Prentice Hall 1999.
- Schmelzer, R et. al. "*XML & Web Services unleashed*", Sams 2002.

- IEEE Xplore (free access from campus) offers IEEE distributed systems and other journals

Slides will be available on the module web page at some point!

Assumptions:

1. You have a decent understanding of TCP/IP networking.
2. You can program! (We'll use C and probably some C++ and Java) and are comfortable with unix derived o.s.'s
3. This is a work in progress! Much may change from what I envisage now.
4. By the end of the module, that you've read CDK!

Assessment:

Coursework – development exercise

Examination – Compulsory question (50%) and Two from Three.

Lab Work:

We'll start by doing some simple socket programming and move on
...and on and on

Broad Outline:

Intro	(Chap. 1, first set of slides)
Client-server	(Chap. 4, socket worksheet)
Remote invocation	(Chap. 5, RPC/RMI worksheets)
Security	(Chap. 7, 2 sets of slides, SSL worksheets)
Naming	(Chap. 9, slides, uPNP(?))
Time	(Chap 10, slides)
Reliability	(Chap 14, slides)

Indicative Only!

What are Distributed Systems?

A very good question! Different authors will have different views.

“A system in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing.” [Coulouris]

“A distributed system is a collection of independent computers that appear to the users of the system as a single computer.” [Tanenbaum]

Reasons for distributing systems

Functional distribution: computers have different functional capabilities.

- Client / server
- Host / terminal
- Data gathering / data processing

sharing of resources with specific functionalities

Inherent distribution stemming from the application domain, e.g.

- cash register and inventory systems for supermarket chains
- computer supported collaborative work

Load distribution / balancing: assign tasks to processors such that the overall

system performance is optimized.

Replication of processing power: independent processors working on the same task

- distributed systems consisting of collections of microcomputers may have processing powers that no supercomputer will ever achieve
- 10000 CPUs, each running at 50 MIPS, yields 500000 MIPS,
 - then instruction to be executed in 0.002 nsec
 - equivalent to light distance of 0.6 mm
 - any processor chip of that size would melt immediately

- Physical separation: systems that rely on the fact that computers are physically separated (e.g., to satisfy reliability requirements).
- Economics: collections of microprocessors offer a better price/performance ratio than large mainframes
 - mainframes: 10 times faster, 1000 times as expensive

Some key points:

Absence of a global clock

–Due to asynchronous message passing there are limits on the precision with which processes in a distributed system can synchronize their clocks

- We'll revisit this in painful detail ☺

Absence of a global state

- No one computer “knows” the state of the “system”

Concurrency

Multithreading, multitasking etc.

Simultaneous execution or access; or the appearance thereof.

Distributed systems need to be concurrent, but concurrency does not imply distributed!

- Necessary but not sufficient!

Parallel Systems

Many authors would regard most parallel systems as distributed.

Flynn, M., *Some Computer Organizations and Their Effectiveness*, IEEE Transactions on Computing, Vol. C-21, p.p. 94, 1972 proposed a taxonomy for computer systems:

Two dimensions depending on instruction and data streams:

- **SISD**
A traditional Von Neuman architecture (e.g. single processor desktop PC) no need to discuss further.
- **SIMD**
original massively parallel machines (e.g. DAP, Connection Machine)

But should we regard these as distributed?

- **MISD**
data flow machine – weird & experimental (Carnegie Melon's c.mmp possibly, or systolic arrays)
- **MIMD**
flexible and common multiprocessor of which there are two subclasses.

These are usually regarded (particularly DM) as distributed systems.

- **MIMD SM - Shared Memory**
very common nowadays
simple to program (mutex, semaphores etc.)
O.S. transparency (how do you know you've more than 1 processor?)
- **MIMD DM - Distributed Memory**
Whilst specialist "one box" parallel computers (e.g. Cray T3D, Meiko computing surface) exist, other distributed systems fall in this box

- require connectivity and communication – issues!

Heterogeneity

Most distributed systems are heterogeneous, that is units consist of different makes, models, performance, technology, processor etc.

- **Build across diverse networks, operating systems, programming languages, computer hardware.**
- **Middleware masks heterogeneity and provides programming abstraction.**
 - **CORBA**
 - **Enterprise Java (limited to Java)**
 - **DCOM (limited to Microsoft systems)**
 - **Mobile Code masks heterogeneity by running virtual machines on all nodes. [Java VM and applets]**

Networking

Distributed systems rely on networks, but distribution implies transparency, that is how many computers are involved for example is hidden.

Networks however rely on distributed systems (e.g. DNS) that run on them to be useful.

Distributed systems in turn need networking functionality.

- Many problems and issues are the same
- Someone has to do some network programming to build a distributed system.

Ubiquitous & Mobile Computing

- Modern networks are becoming “ad-hoc”
- Bigger more complex networks are being built (e.g. cellular phones)
- Devices are mobile – location based services

Openness

At on level (FLOSS) openness may simply be regarded as A Good Thing!

At another we need to make media/services/content available to heterogeneous hosts.

- Publish documents and specifications.
 - Internet RFC's (IETF), CORBA specifications (OMG), W3C specifications
- Publish interfaces
 - specifications
 - dynamic meta-modeling and reflection

Transparency

The system is perceived as working as whole rather than a collection of independent components. Gives rise to perceived wholeness properties of the system

ISO Reference Model for Open Distributed Processing identifies the following transparencies.

- **location & access**
 - **concurrency**
 - **replication**
 - **failure**
 - **performance**
 - **mobility**
 - **scalability**
-
- *Access transparency*: enables local and remote resources to be accessed using identical operations.
 - *Location transparency*: enables resources to be accessed without knowledge of their location.
 - *Concurrency transparency*: enables several processes to operate concurrently using shared resources without interference between them.
 - *Replication transparency*: enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers.

- *Failure transparency*: enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components.
- *Mobility transparency*: allows the movement of resources and clients within a system without affecting the operation of users or programs.
- *Performance transparency*: allows the system to be reconfigured to improve performance as loads vary.
- *Scaling transparency*: allows the system and applications to expand in scale without change to the system structure or the application algorithms.

Security

Many distributed systems in an exploitable environment (e.g. internet banking). Security (as well as authentication and privacy) come from cryptographic processes.

We'll meet this again in painful detail 😊

Scalability

- it works for 100's will it work for 1000's and 1000000's
- can the system grow over a larger geographical extent & range without degradation of performance.

Some Examples

Internet:

- Routing protocols (SPF, vector distance, RIP)
- Name service (Domain Name Service)
- Time Service (Network Time Protocol NTP)
- Applications (ftp, email, telnet, SNMP, ...)
- Resource Sharing & the Web
- NFS [a very transparent Network File System]
- Work Flow Management systems
- WWW [resource publishing & access service]

Mobile & Ubiquitous Computing

- adds a new dimension to distributed systems with new devices[PDA's, phones, cameras, watches, domestic appliances, wearable computers, GPS]
- Also new protocols/standards HAVi, uPNP

A working definition:

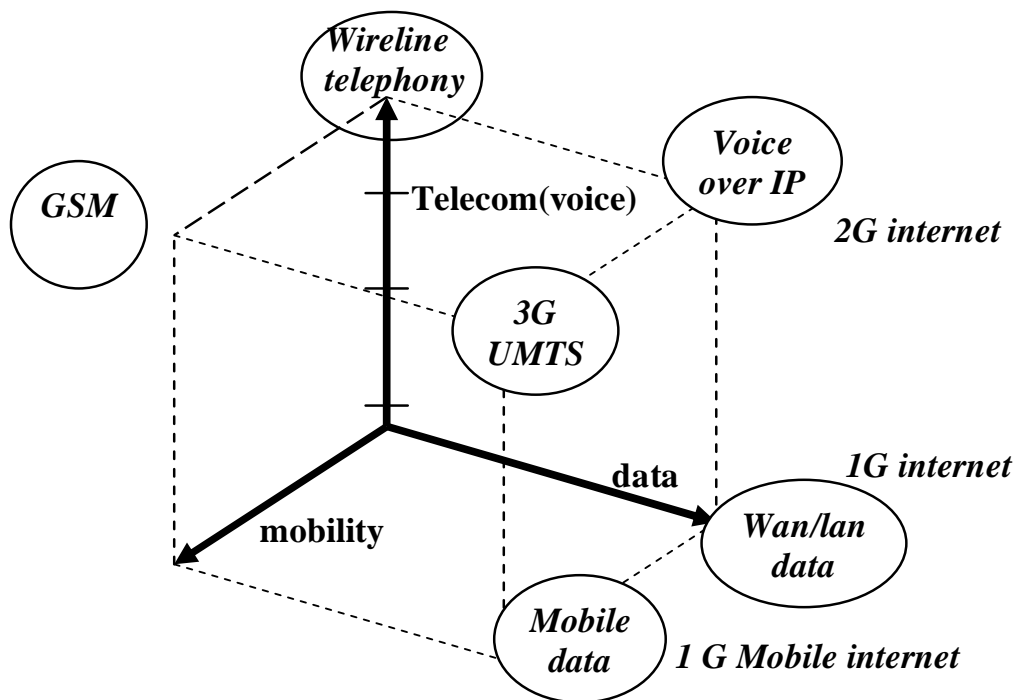
- **A distributed system is one in which hardware and software components located at network nodes co-ordinate and communicate their actions by message passing.**

or

- **is a system in which some remote networked object that you have never ever heard of crashes the computer you are working on.**

Internet, Telecoms and Mobile convergence

- Distributed systems we build work over networks.
- Today single service networks
 - *Radio/TV,*
 - *Fixed telephony,*
 - *GSM,*
 - *WAN & LAN(data networks)*
- Future is multi-service networks
- Cerf's Prediction
 - *By 2010 100% of all traffic will be packetized*
 - Skype (VOIP)
 - 3G cellular



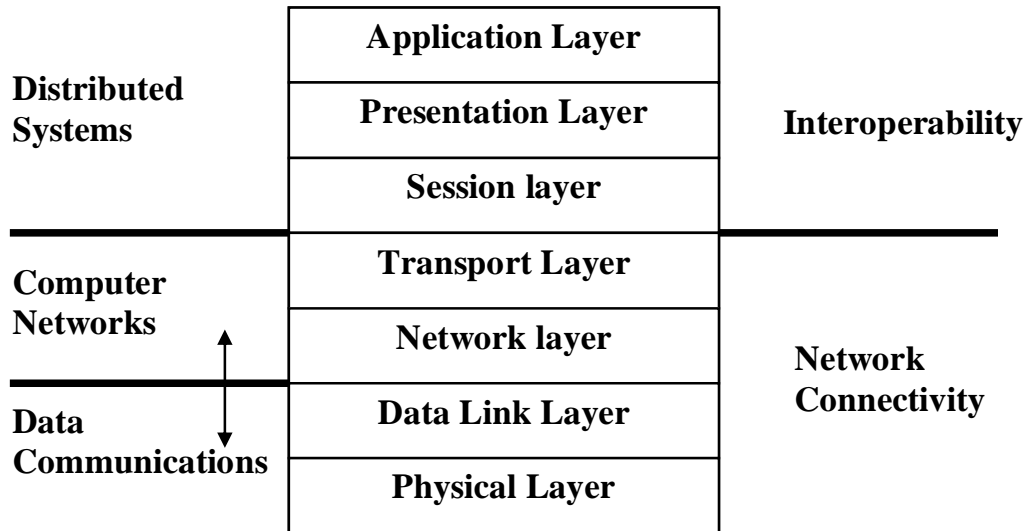
Technology enablers

- *WAP / I-mode*
- *MPEG4*
- *WinCE, Java, Symbian, PalmOS, Linux*
- *GPS – Location based services & presence*
- *Ipv6*
- *Bluetooth*
- *IMT2000 - Umts (WCDMA)*

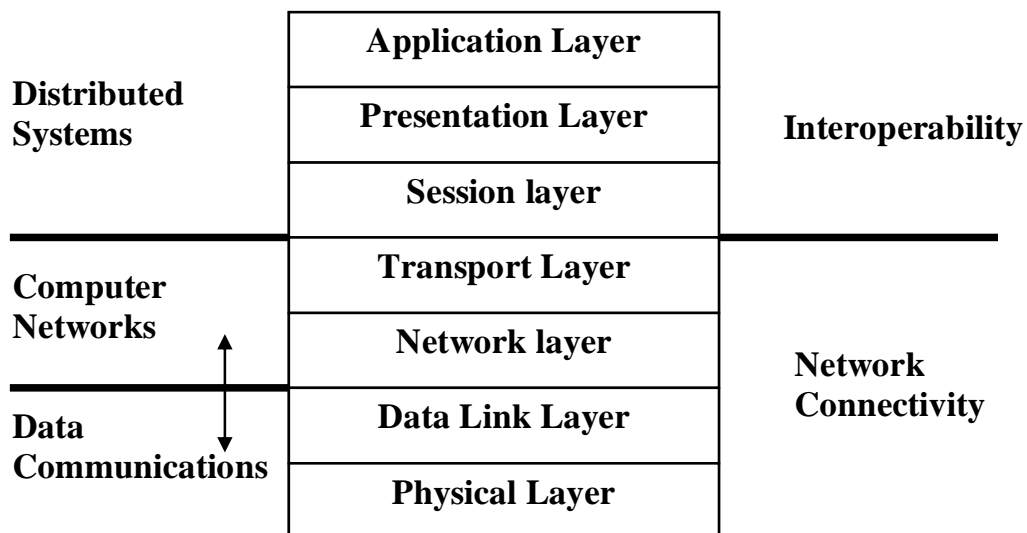
Models and Architectures for Distributed Systems

- **Client Server & Distributed Object Models**
 - *Tightly coupled with well defined interfaces & protocols, standards*
 - *(Sockets [Internet Applications], RPCs [NFS],*
 - *Object Models [OMG OMA, Microsoft DCOM]*
- **JINI**
 - *Resource discovery, flexible, tuple space, event based*
- **WWW**
 - *Document based, loosely coupled with lots of adaptors & plug-ins*
- **Semantic Web & GRIDs (Next Generation)**
 - *Federated: collaborate by describing using meta-data, diversity of standards*
- **Bluetooth/Jini/uPNP discovery & advertisement**

Distributed Systems are built on networks!

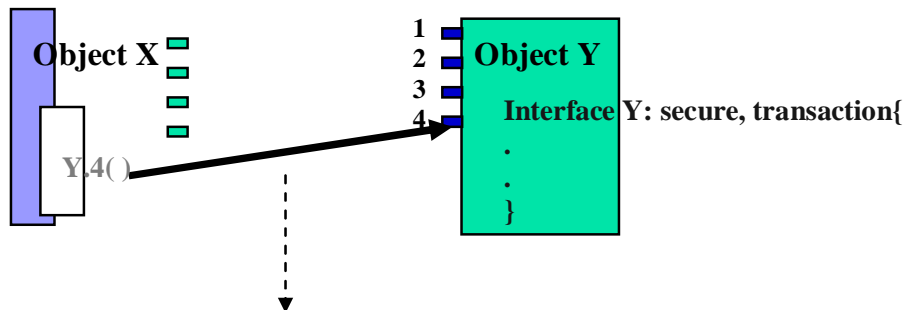


Middleware masks complexity & heterogeneity



Middleware

- Often based on a DOM (Distributed Object Model)
- Structured model with well defined component interfaces or API's to manage the complexity.
- Well defined interfaces provide a contract between service provider and client.
- Fundamental distributed services must appear transparent to the application builder. (Most distributed services are beyond the average programmers ability to build.)
- Encapsulation property of objects hides heterogeneity and helps manage complexity by forming units of:-
 - service, distribution, failure, security
- Object interface provides service and service contract to clients. Allows use of multiple implementation languages.
- Objects may be organized as a class hierarchy according to services provided.



Fundamental services are transparent to application developed objects which are hidden behind the invocation call. So the call from object X to Y is a transactional & secure method call. The developer defines Y to be of interface class secure and transaction (multiple inheritance).

Example DOM architectures:

- **OMG OMA CORBA**
- **Microsoft DCOM/COM+**
- **Microsoft .NET (object based)**
- **Enterprise Java**
- **Jini**