DS 2010 introduction

David Evans
de239@cl.cam.ac.uk

Distributed systems, Easter 2010

1. **Introduction** system, legal, social context; technology-driven evolution; fundamental characteristics; software structure; models, architecture, engineering;
2. **Time** event ordering; physical clock synchronisation; process groups; ordering message delivery;
3. **Distributed algorithms and protocols** strong and weak consistency, replicas; concurrency control; atomic commitment; election algorithms; distributed mutual exclusion;
4. **Middleware** RPC, OOM, MOM, event-based middleware;
5. **Naming**
6. **Access control** capabilities, ACLs, RBAC and access control policy; OASIS RBAC case study;
7. **Storage services** distribution issues
8. **Event-driven systems**

How to think about distributed systems

Life is grim

Fundamental characteristics

1. concurrent execution of components
2. independent failure modes
3. transmission delay
4. no global time

What this course is *not* about

- distributed operating systems
- cloud computing

Distributed systems: introduction

- some systems background
- some legal & social context
- development of technology—DS evolution
- implications of the fundamental characteristics
- software structure for a node
- model, architecture, engineering
- architectures for doing it on a large scale

Costly failures

- UK stock exchange share trading system
  - abandoned 1993, £400M
- California automated childcare support
  - suspended 1997, $300M
- US tax system modernisation
  - scrapped 1997, $\$4 \times 10^9$
- UK ASSIST, statistics on welfare benefits
  - terminated 1994, £3.5M
- London Ambulance Service computer-aided despatching
  - scrapped 1992, £7.5M, 20 deaths in 2 days

What makes things special?

- normal software failure

⇒ errant behaviour not accommodated by other parts of the system

⇒ a cascade of the failure that is spectacular

Where do high expectations come from?

- Web experience (though some of these are DS!)
  - *e.g.* information services: trains, postcodes, …
  - *e.g.* online banking
  - *e.g.* airline reservations
  - *e.g.* conference management
  - *e.g.* online shopping
- Things mostly work, helped by
  - read mostly
  - client-server
  - closely coupled
  - synchronous interaction (request-reply)
  - single-purpose
  - (often) private sector
  - (often) focussed

Public-sector systems especially…

- are bespoke and complex
- are large
- have heterogeneous clients and roles
- need a web portal, but are not like “traditional” web sites
- must be around for a long time
- often have ubiquitous/mobile requirements
- are influenced by competition & independent procurement vs. interoperation
- are influenced by legislation and government policy

You encounter DS all the time

- CDNs (Akamai, . . .)
- Facebook
- any IM service

. . . but incorrect behaviour isn't too deleterious. Why?

Some legal and policy requirements

- exclusions: “patients may specify who may see, and not see, their electronic health records (EHRs)”

Some legal and policy requirements

- exclusions: "patients may specify who may see, and not see, their electronic health records (EHRs)"
- relationships: "only the doctor with whom the patient is registered (for treatment) may prescribe drugs, read the patient's EHR, *etc.*"

Some legal and policy requirements

- exclusions: "patients may specify who may see, and not see, their electronic health records (EHRs)"
- relationships: "only the doctor with whom the patient is registered (for treatment) may prescribe drugs, read the patient's EHR, *etc.*"
- "the existence of certain sensitive components of EHRs must be invisible, except to explicitly authorised roles"

Some legal and policy requirements

- exclusions: "patients may specify who may see, and not see, their electronic health records (EHRs)"
- relationships: "only the doctor with whom the patient is registered (for treatment) may prescribe drugs, read the patient's EHR, *etc.*"
- "the existence of certain sensitive components of EHRs must be invisible, except to explicitly authorised roles"
- "buses should run to time and bus operators will be punished if published timetables are not met", so bus operators can be reluctant to cooperate in traffic monitoring, even though monitoring could show that delay is often not their fault

Data protection legislation

Gathered data that identify individuals must not be stored.

- CCTV camera software must not recognise people and store identities with images
- vehicle number plates must not be recognised and then linked to and stored with identities
- Complications from the FOIA

Very messy and changes constantly.

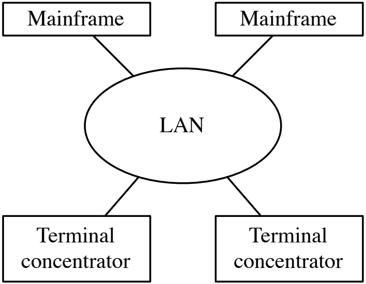
Rapid development and new technology

- don’t get to build a second system
- rapid obsolescence means that incremental growth isn’t sustainable…
- …but should design for incremental deployment
- may demand use by mobile workers (healthcare, police, utilities) and include cameras/sensors

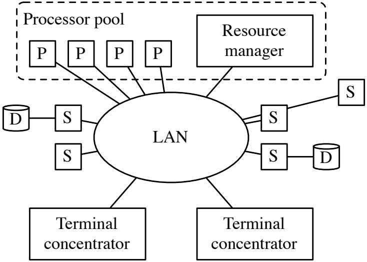
DS history: technology-driven evolution

- fast, reliable LANs (*e.g.*, Ethernet, Cambridge Ring) made DS possible in the 1980s
- early research was on distribution of OS functionality
  1. terminals + multi-access system(s)
  2. terminals + pool of processors + dedicated servers (Cambridge CDCS)
  3. diskless workstations + servers (Stanford)
  4. workstations + servers (Xerox PARC, MIT Athena)
- now WANs are fast and reliable, so. . .

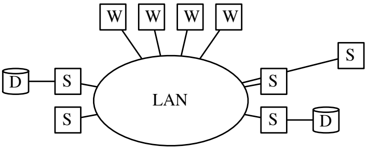
terminals + multi-access system(s)



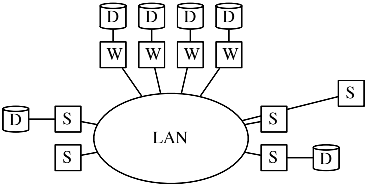
terminals + processor pool + servers



diskless workstations + servers



workstations + servers



The evolution of comms.

- WANs quickly became as high bandwidth and reliable as LANs
- new problems, such as flash crowds
- bandwidth has become high, but latency was and remains a problem, partially due to end-system processing time for huge numbers of clients

Fundamental characteristics

1. concurrent execution of components
2. independent failure modes
3. transmission delay
4. no global time

Implications:

- 2, 3 can't know why there's no reply—node/comms. failure and/or node/comms. congestion
- 4 can't use locally generated timestamps for ordering distributed events
- 1, 3 inconsistent views of state/data when it's distributed
- 1 can't wait for quiescence to resolve inconsistencies

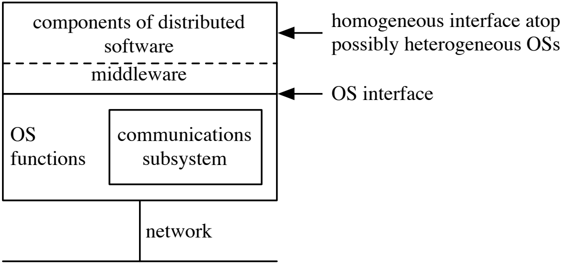
These determine

- software structure for a node
- model/architecture/engineering for a system

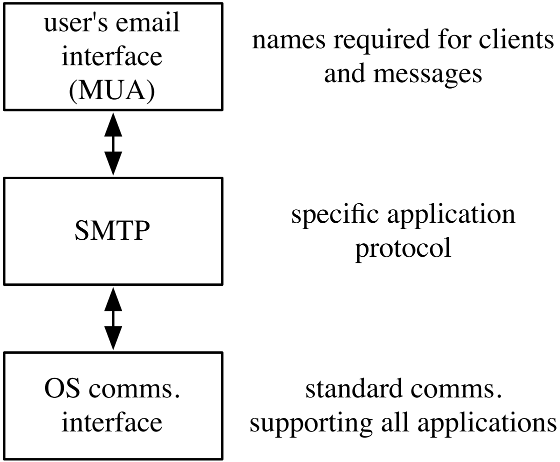
Single node software structure

Support for distributed software may be

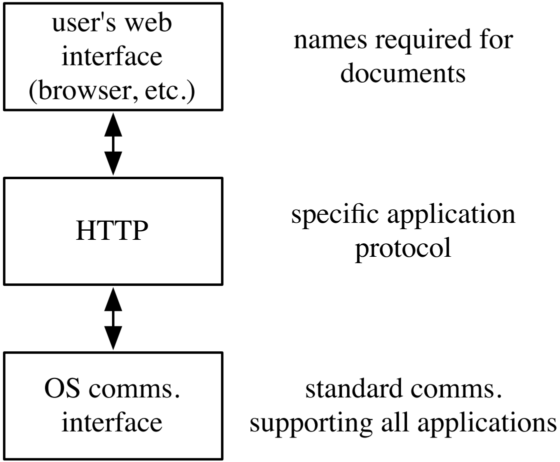
1. directly by OS in a homogeneousish cluster (distributed OS design)—not the focus of this course
2. by a software layer (middleware) above one or more potentially heterogeneous OSs



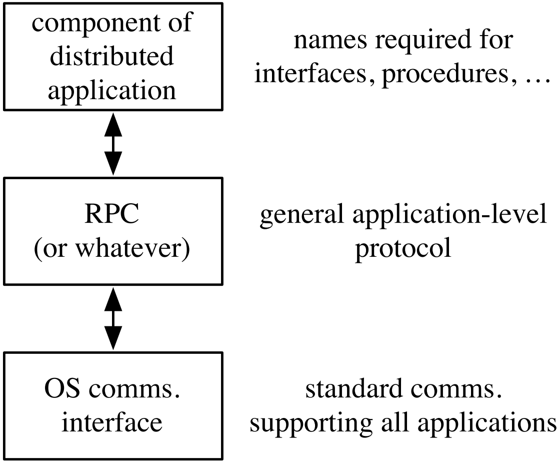
Distributed application structure: email, ftp, . . .



Distributed application structure: the web



Distributed application structure: in general



Open and proprietary middleware

- evolution controlled by standards bodies or consortia
- interoperability “bake-offs” are not uncommon
- resulting system something of a compromise, which can be good or bad

versus

- can be changed by the owner (users may need to buy a new release)
- consistency across versions is not guaranteed
- good for technical extortion

Interoperability and languages

- can your system span multiple middlewares (including different implementations of the same MW)?
- can components be written in different languages and interoperate?

Model of distributed computation

- what are the named entities? objects, components, services, . . .

Model of distributed computation

- what are the named entities? objects, components, services, . . .
- how is communication achieved?
    - synchronous/blocking (request-response) invocation, *e.g.*, the client-server model
    - asynchronous messages, *e.g.*, the event notification model
    - one-to-one, one-to-many?

Model of distributed computation

- what are the named entities? objects, components, services, ...
- how is communication achieved?
  - synchronous/blocking (request-response) invocation, *e.g.*, the client-server model
  - asynchronous messages, *e.g.*, the event notification model
  - one-to-one, one-to-many?
- are the communicating entities closely or loosely coupled?
  - must they share a programming context?
  - must they be running at the same time?

System architecture

…the framework within which the entities in the model interoperate

- naming
- location of named objects
- security of communication
- authentication of participants
- protection/access control/authorisation
- replication to meet requirements for reliability, availability

Entities may be defined within *administration* domains; need to consider multi-domain systems and interoperation within and between domains

System engineering implementation decisions

- placement of functionality: client libraries, user agents, servers, wrappers/interception
- replication for failure tolerance, performance, load balancing ⇒ consistency issues
- optimisations, *e.g.*, caching, batching
- selection of standards, *e.g.*, XML, X.509
- what "transparencies"[1] to provide at what level
  - distribution transparency: location? failure? migration?

  ⇒ may not be achievable or may be too costly

[1] hidden from application developer; needn't be programmed for, can't be detected when running

Architectures for large-scale, networked systems

1. federated administration domains
   - integration of databases
   - integration of sensor networks
   - small dynamic domains with members grounded in various static administration domains
2. independent, external services to be integrated
3. detached, ad hoc, anonymous groups; anonymous principals, issues of risk and trust

Examples of federated administration domains

- **national healthcare services**: many hospitals, clinics, primary care practices
- **national police services**: county police forces
- **global company**: branches in London, Tokyo, New York, Berlin, Paris, ...
- **transport**: County Councils responsible for cities, some roads
- **active city**: fire, police, ambulance, healthcare services; mobile workers; sensor networks, *e.g.*, for traffic/pollution monitoring

Federated domains—characteristics

- names administered per domain (users, roles, services, data-types, messages, sensors, . . .)
- authentication users administered within a domain
- communication needed within and between domains
- security per-domain firewall-protection
- policies specified per domain, *e.g.*, for access control; *intra-* and *inter-domain*, plus some external policies to satisfy government, legal and institutional requirements
- high trust high accountability

Small dynamic domains with members grounded in static administration domains

- *e.g.*assisted home-living (sheltered housing) “patient” + various carers + technology
- carers have roles in primary care practices, hospitals, social services, out-sourced services
- care programme is specified by contract
  - rights of patient to defined care
  - obligations of carers and patients
  - privacy of patient data
  - need to audit people and technology

Dynamic domains—characteristics

- names principals (users, roles): from home domains; services, data types, messages, sensors set up for small dynamic domains
- authentication users administered within home domain; need for credential check back to home domain (as in federated domains)
- communication needed across domains
- policies indicate contractual obligations and privileges (access control)
- audit of people, technology
- trust based on observation of audit (and reputation?)

Examples of independent, external services

- **commercial web-based services**: online banking, airline booking
- **national services used by police and others**: DVLA, court-case workflow
- **national health services**: national Electronic Health Record (EHR) service
- **e-science (grid) databases and generic services**: astronomical, transport, medical databases for computation or storage
- **virtual organisations**: collaborating groups across several domains

Independent, external services—characteristics

- naming and authentication may be client-domain-related, and/or of individuals via certification authorities (CAs)
- policies related to client roles in domains and/or individual principals may provide support for "virtual organisations"
- need accounting, charging, audit
- trust based on evidence of behaviour; clients exchange experiences, services monitor and record; often assume full connectivity

Detached, ad hoc, anonymous groups

- may be connected by wireless
- can't assume trusted third parties (CAs) accessible
- can't assume knowledge of names and roles; identity likely to be by key/pseudonym
- new identities can be generated (by detected villains)
- parties need to decide whether to interact
- each has a trust policy and a trust engine
- each computes whether to proceed—policy is based on:
  - accumulated trust information (from recommendations and evidence from monitoring)
  - risk (resource-cost) and likelihood of possible outcomes

Examples of detached, ad hoc, anonymous groups

- Commuters regularly play cards on the train
- E-purse purchases
- Recommendations of people and, *e.g.*, restaurants in a tourist scenario
- Wireless routing via peers
- Routing of messages P2P rather than by dedicated brokers—reliability, confidentiality, altruism
- Trust has a context

Promising approaches for large-scale systems

- Roles for scalability
- Parametrised roles for expressiveness
- RBAC for services, service-managed objects, including the communication service
- Policy specification and change management
- Policy-driven system management
- Asynchronous, loosely-coupled communication: publish/subscribe for scalability; event-driven paradigm for ubiquitous computing
- Database integration—how best to achieve it?

The OPERA group—research themes

Some are specific projects (from the past or present); some are principles.

- Access Control: OASIS RBAC (Open Architecture for Securely Interworking Services)
- Policy expression and management
- Event-driven systems: CEA, Hermes, EDSAC21
- Trust and risk in global computing (EU SECURE): secure collaboration among ubiquitous roaming entities
- TIME: a Traffic Information Monitoring Environment
  - TIME-EACM Event Architecture and Context Management
- CareGrid: dynamic trust domains for healthcare applications
- SmartFlow: extensible event-based middleware

The OPERA group—research themes

see
http://www.cl.cam.ac.uk/Research/SRG/opera
for people, projects, publications for download