

Distributed Systems - Overview

- some systems background/context
- some legal/social context
- development of technology – DS evolution
- **** DS fundamental characteristics ****
- software structure for a node
- model/architecture/engineering for a DS
- architectures for large-scale DS
 - federated administration domains
 - integrated domain-independent services
 - detached, ad-hoc groups

Costly Failures in Large-Scale Systems

- **UK Stock Exchange** - share trading system
 - abandoned 1993, cost £400M
- **CA automated childcare support**
 - pended 1997, cost \$300M
- **US tax system** modernisation
 - scrapped 1997, cost \$4B
- **UK ASSIST**, statistics on welfare benefits
 - terminated 1994, cost £3.5M
- **London Ambulance Service Computer Aided Despatching (LASCAD)** scrapped 1992, cost £7.5M, 20 lives lost in 2 days

Why high public expectation?

Web experience

e.g. information services: trains, postcodes, phone numbers

e.g. online banking

e.g. airline reservation

e.g. conference management

e.g. online shopping and auction

Properties: read mostly, server model, client-server paradigm, closely coupled, synchronous interaction (request-reply), single-purpose, (often) private sector

Public-Sector Systems

healthcare, police, social services, immigration, passports, DVLA (driver + vehicle licensing), court-case workflow, tax, independent living for the aged and disabled, ...

- bespoke and complex
- large scale
- many types of client (many roles)
- web portal interface, but often not web-service model
- long timescale, high cost
- ubiquitous and mobile computing – still under research
- former policy of competition and independent procurement
- current policy requiring interoperation
- legislation and government policy

Some Legal/Policy Requirements - 1

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

“only the doctor with whom the patient is registered (for treatment) may e.g. prescribe drugs, read the patient’s EHR, etc.” - relationships

“the existence of certain sensitive components of EHRs must be invisible, except to explicitly authorised roles”

Some Legal/Policy Requirements - 2

“buses should run to time and bus operators will be punished if published timetables are not met.”

so bus operators refuse to cooperate in traffic monitoring, even though monitoring could show that delay is often not their fault.

Data Protection Legislation

Gathered data that identifies individuals must not be stored:

CCTV cameras: software must not *recognise* people and store identities with images

(thermal imaging (infra-red) - just monitor/count)

Vehicle number plate recognition: must not be associated with people then stored with identities

(only police allowed to look up)

Police records: accusations that are not upheld?

Sally Geeson murder - previous army records of LC Atkinson

Soham murders – previous police records of Huntley;

Govt. now require interaction between counties

UK Freedom of Information Act: Jan 2005

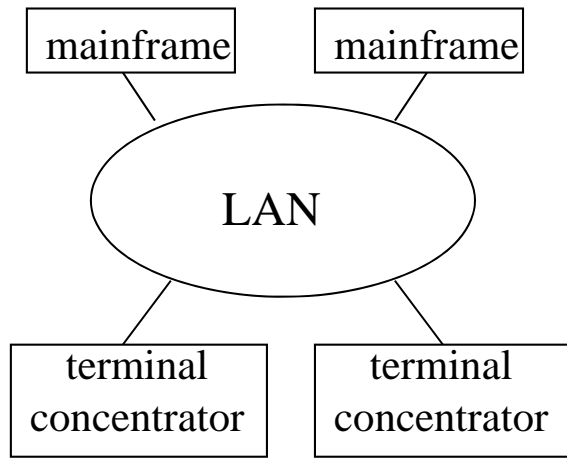
Rapidly Developing and New Technology

- can't ever design a “*second system*”, it's always possible to do more next time
- rapid obsolescence - *incremental growth* not sustainable long-term (unlike e.g. telephone system)
 - a current software engineering research area
- but *big-bang* deployment is a bad idea
design for *incremental deployment*
- *mobile* workers in healthcare, police, utilities etc.
 - integration of wired and wireless networks
- ubiquitous computing: integration of *camera* and *sensor* data

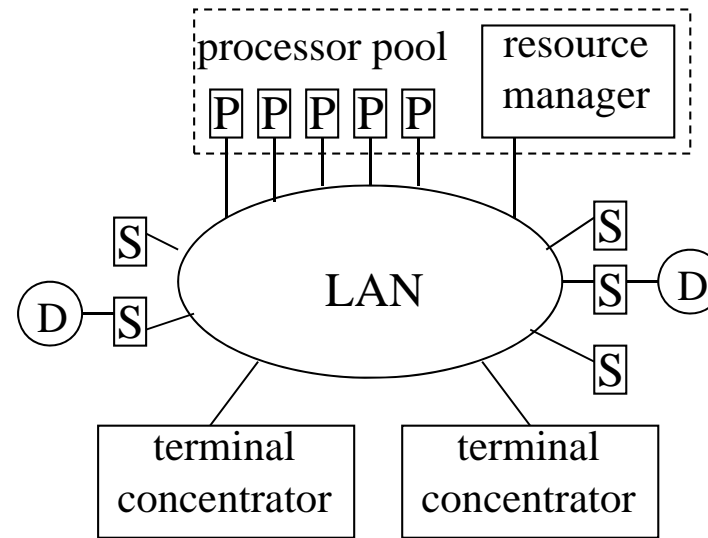
DS history: technology-driven evolution

- Fast, reliable (interconnected) LANs (e.g. Ethernet, Cambridge Ring) made DS possible in 1980s
- Early research was on distribution of OS functionality
 1. terminals + multiaccess systems
 2. terminals + pool of processors + dedicated servers (Cambridge CDCS)
 3. Diskless workstations + servers (Stanford)
 4. Workstations + servers (Xerox PARC)
- Now WANs are fast and reliable,

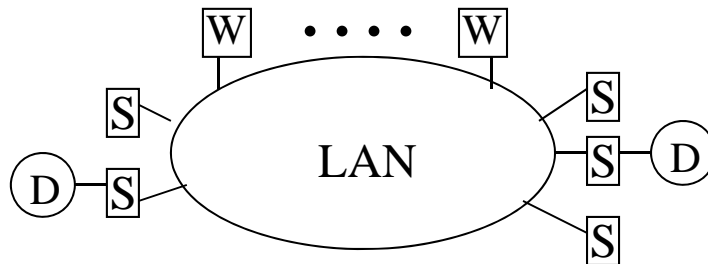
1. LAN as terminal switch to multiaccess systems



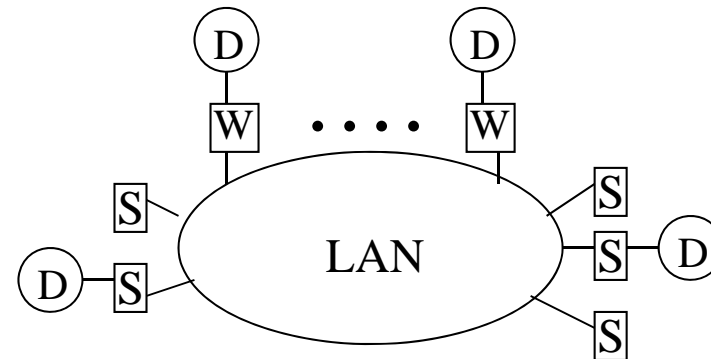
2 terminals + processor pool + servers



3 diskless workstations + servers



4 workstations + servers



Technology-driven evolution – comms.

- WANs *quickly* became as high bandwidth and reliable as LANs
- Distributed database research such as *data-shipping -vs- query-shipping* became obsolete in the 1990s
- *Web services* created new problems – flash crowds
- Bandwidth had become high, but *latency* was and remains a problem, due to end-system processing time for huge numbers of clients
- See lecture DS-7: content storage and delivery for web servers

How to think about Distributed Systems

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

DS fundamental characteristics

1. Concurrent execution of components
2. Independent failure modes
3. Transmission delay
4. No global time

Implications:

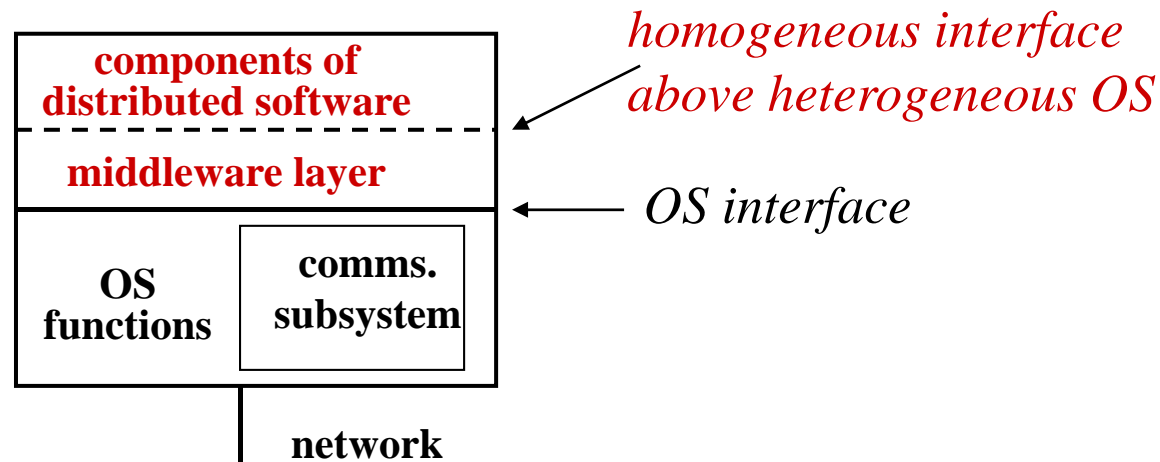
- 2, 3 - can't know why there's no reply – node/comms. failure and/or node/comms. congestion
- 4 - can't use locally generated timestamps for ordering distributed events
- 1, 3 - inconsistent views of state/data when it's distributed
- 1 - can't wait for quiescence to resolve inconsistencies

single node - software structure

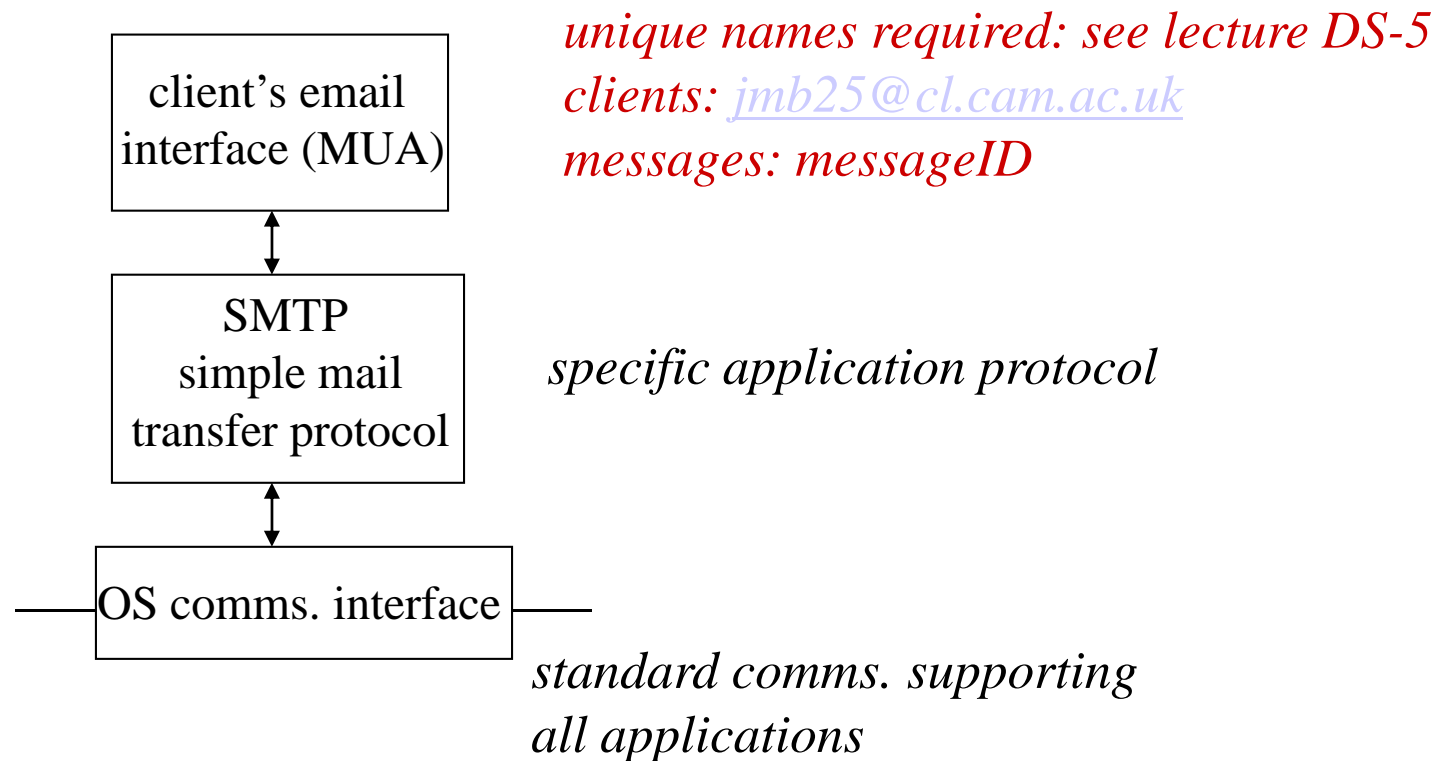
Support for distributed software may be:

special case: directly by OS in a *homogeneous* cluster (distributed OS design) – not the focus of this course

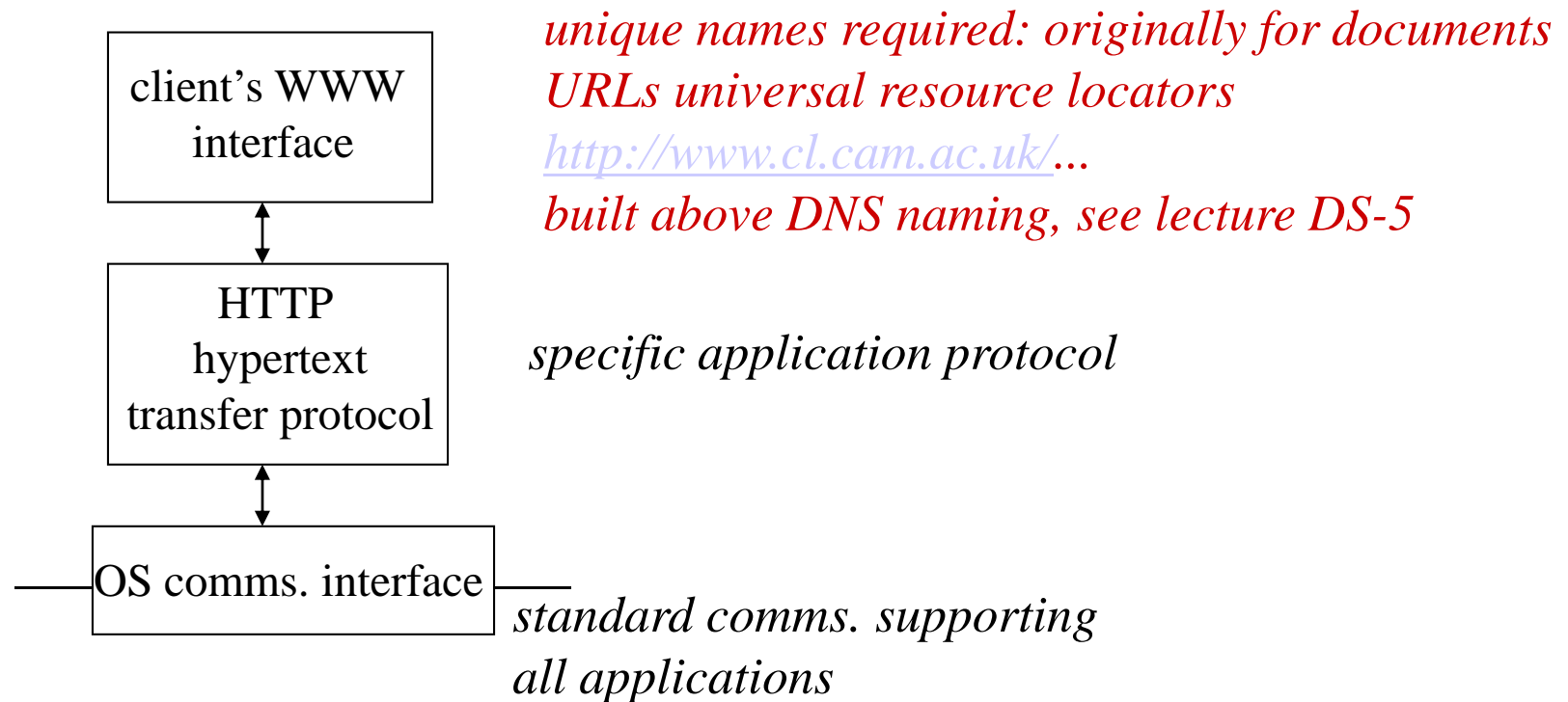
general case: by a software layer (middleware) above potentially heterogeneous OS



Distributed application structure – email, news, ftp



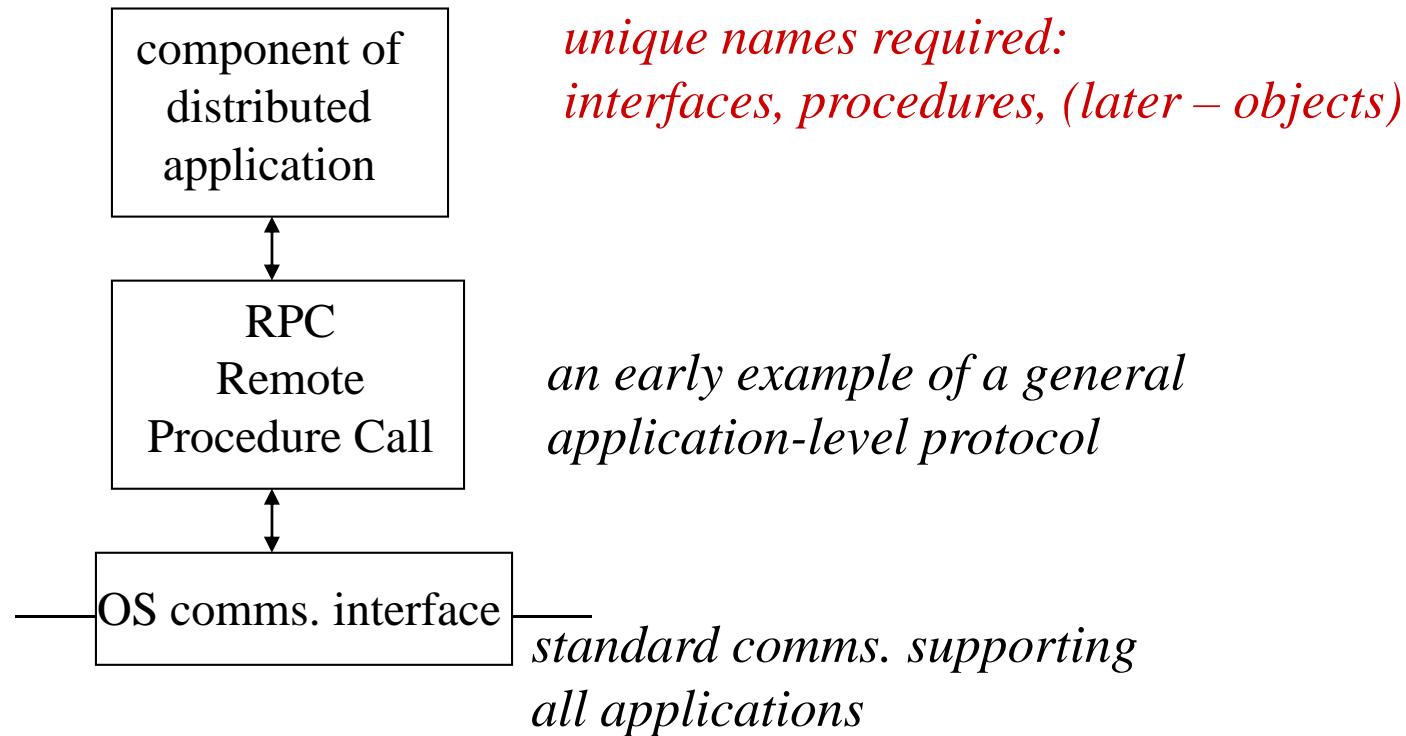
Distributed application structure – web documents



A browser interface came to be used for **general** distributed applications

W3C standards for **Web Services** – see lectures DS-4, DS-7

Distributed application structure – general support example



RPC is an early example of a protocol above which distributed applications may be developed. RPC examples: ISO-ODP, OSF-DCE
A middleware also includes [services](#) above the RPC layer

Open and proprietary middleware

- **Open:** evolution is controlled by standards bodies (e.g. ISO, NIST) or consortia (e.g. OMG, W3C). Requests for proposals (RFPs) are issued, draft specifications published with RFCs (requests for comments). Compromise is common.
- **Closed, proprietary:** can be changed by the owner (clients need to buy a new release). Consistency across versions is not guaranteed. Plus “*embrace, enhance, extinguish*”.

Related issues:

- **single/multi language:** can components be written in different languages and interoperate?
- **open interoperability:** is desirable across middlewares (including different implementations of the same MW)

DS Design: model, architecture, engineering

Programming **model** of distributed computation:

- What are the named entities? objects, components, services,...
- How is communication achieved?
 - synchronous/blocking (request-response) invocation
e.g. client-server model
 - asynchronous messages e.g. 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, as required by applications
- Authentication of participants
- Access control / authorisation
- Replication to meet requirements for reliability, availability

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*” to provide at what level:
(transparent = hidden from application developer: needn't be programmed for, can't be detected when running).
distribution transparency: location? failure? migration?
may not be achievable or may be too costly

Architectures for Large-Scale, Networked Systems

Individual user using globally available service

Single administration domain

Federated administration domains

Independent, external services - to be integrated

Detached, ad-hoc, anonymous groups;
anonymous principals, issues of trust and risk

Federated administration domains: Examples

- **national healthcare services:**
many hospitals, clinics, primary care practices.
- **national police services:**
43 county police forces (52 with Scotland),
- **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 **communication**, **access control** *intra and inter-domain*, plus some external policies to satisfy government, legal and institutional requirements
- **high trust** and accountability within a domain,
known trust between domains

Independent, External Services - Examples

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

Independent, external services - characteristics

- **naming and authentication**
may be of individuals via trusted third parties (TTPs)
and/or via home domain of client
- **access control policies**
related to client roles in domains and/or individuals
support for “*virtual organisations*” spanning domains
- need for: **accounting, charging, audit**
these may be done by trusted third parties
a basis for mutual **trust** (service done, client paid)
- **trust**
based on evidence of behaviour
clients exchange experiences, services monitor and record
assume full connectivity, e.g. TTPs can authenticate/identify

Examples of detached ad hoc groups and the need for trust

- Commuters regularly play cards on the train
- Auctions – build up trust of an ID through small honoured purchases, then default on a big one
- E-purse purchases – trust in system
- Recommendations: e.g. in a tourist scenario - restaurants, places to visit. Recommendations of people and their skills.
- Wireless routing via peers:
routing of messages P2P rather than by dedicated brokers –
reliability, confidentiality, altruism
- Trust has a context – skills may not transfer
e.g. drivers of cars, trains, planes ...

3. Detached, ad-hoc, anonymous groups

- e.g. 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

Promising Approaches for Large-Scale Systems

- **Roles** for scalability
- **Parametrised roles** for expressiveness, scalability, simplicity
- **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?

And don't forget:

- **Mobile** users
- **Sensor network** integration

Opera Group – research themes

(**objects policy events roles access control**)

- Access Control (**OASIS** RBAC)
Open **A**rchitecture for **S**ecurely **I**nterworking **S**ervices
- **P**olicy expression and management
- Event-driven systems (**CEA, Hermes**)
EDSAC21: event-driven, secure application control for the 21st Century
- Trust and risk in global computing (EU **SECURE**)
secure collaboration among ubiquitous roaming entities
- **TIME**: Traffic Information Monitoring Environment
TIME-EACM event architecture and context management
- **CareGrid**: dynamic trust domains for healthcare applications
- **SmartFlow**: Extendable event-based middleware
- **PAL**: personal and social communications services for health and lifestyle monitoring.

see: www.cl.cam.ac.uk/research/srg/opera

for people, projects, publications for download