Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

APM

Abstract and Automate (Object World UK 1995 presentation)

Andrew Watson

Abstract

Distributed object infrastructures that span organisational boundaries promise great benefits in terms of reduced costs and increased productivity in the organisations they serve. On the other hand, they are complex and difficult to build, and subsequently maintain, in the face of rapid technological and organisational change. Attempts to control this complexity by enforcing homogeneity are doomed to failure. The only hope is to design using high levels of abstraction and automate the construction where-ever we can.

This set of colour slides was written for a talk at Object World UK 1995. The title and some of the content is the same as RC.418, but with the addition of material explaining why industry is increasingly moving towards use of distributed systems.

APM.1497.01 **Approved** 1st June 1995
External Paper

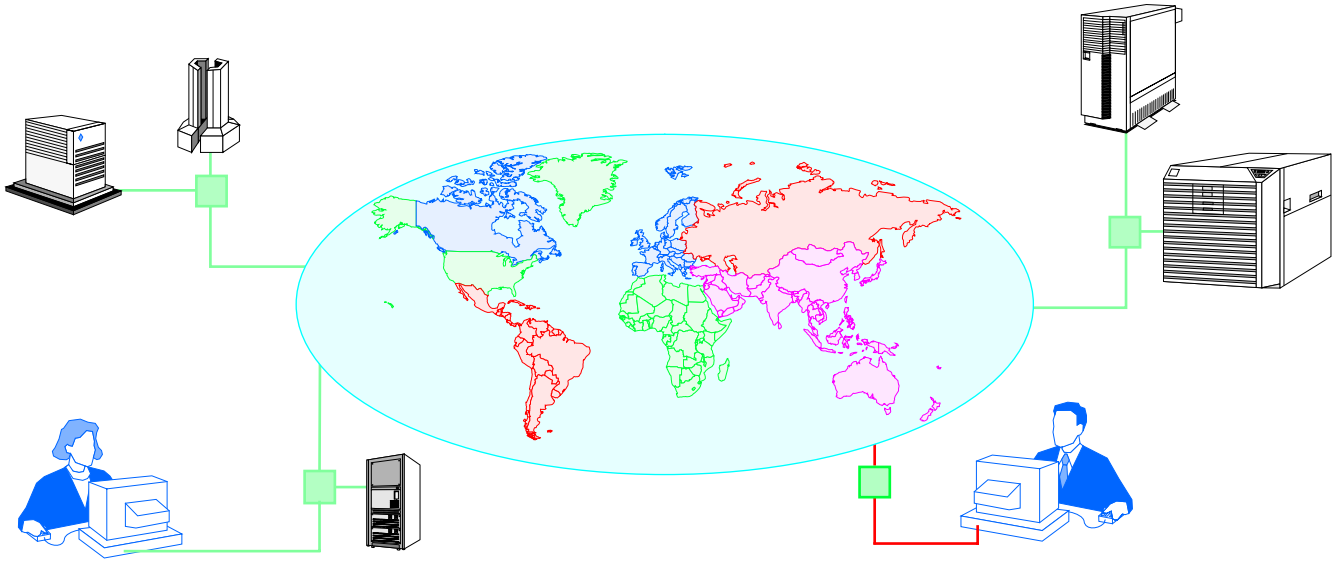
Distribution:
Supersedes:
Superseded by:

Abstract and Automate

Andrew Watson
APM
ajw@ansa.co.uk

The Opportunity

- **We are currently building a world-wide computer system**
  - **This will be the largest machine ever built**
  - **It will encompass and dwarf the international telephone network**



The Challenge

- The total costs will be enormous
- We'll only ever build one
- Nothing this complex has been attempted before
- Nobody is in charge
  - Many participants don't realise they are involved
  - There is no overall design
- The technology is very diverse
  - and evolving faster than it can be deployed
- When we've built it, can we maintain it?
  - ... and continue to reap the benefits

Monolithic systems once ruled

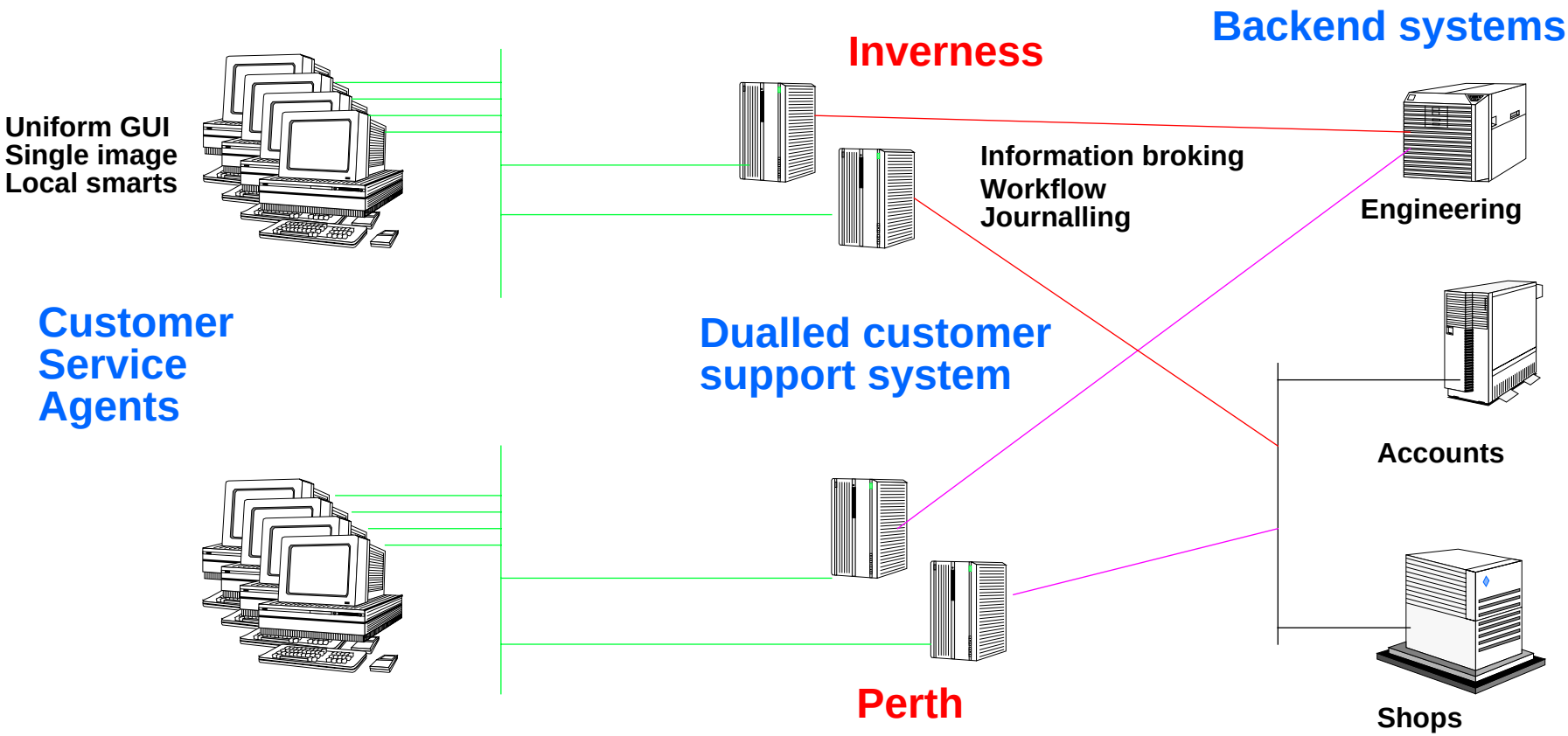
- **Distributed systems played a minor role in commercial DP in the past**

- **Hard to design and implement**
  - **You don't want to do this unless you have no choice**
  - **Centralised systems usually cheaper to build and maintain**
  - **... even if that means changing the whole organisation around them**
  - **e.g. High Street banks**

The times, they are a'changing

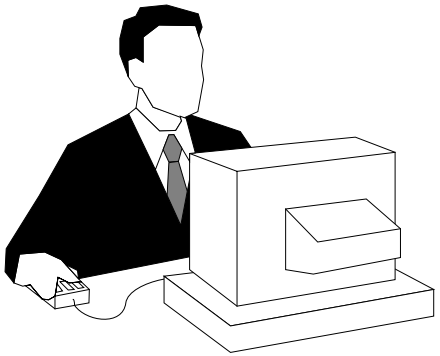
- With internal automation, larger proportions of costs and delays occur at organisational interfaces
  - My stock-control system prints out an order which I send to you, so you can type it into your order-processing package
  - It's laborious, relatively slow, and expensive
  - Economic imperative towards electronic commerce and EDI
  - 6,200 British companies (including 9 of the top 10 supermarkets) use INS
- Using people to "glue" together legacy systems inside one organisation also inefficient

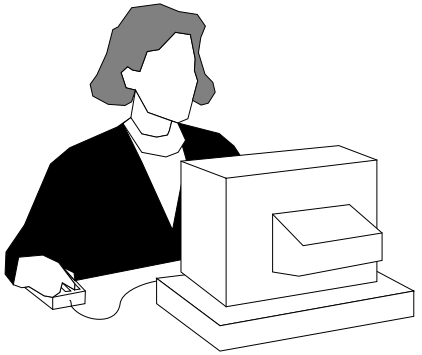
An Example - Scottish HYDRO



New business opportunities

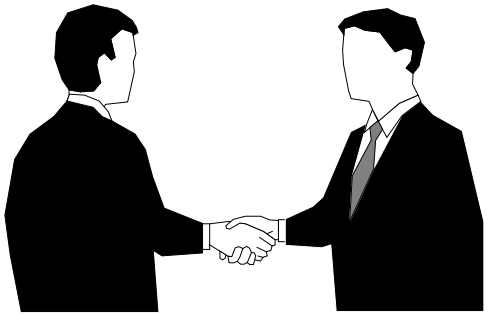
- **Computer networking and electronic commerce also open up new business opportunities**
  - **Volvo now build every car to specification for individual customers**
  - **Investment trusts use BACS to invest small sums for thousands of investors**

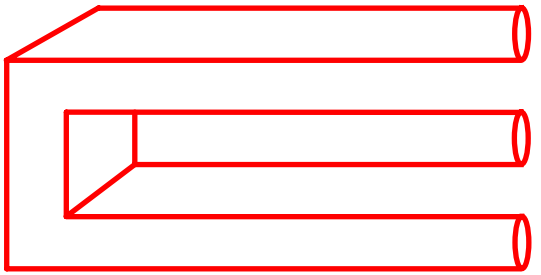




The challenges

- **New challenges of trans-organisational computing**
  - **Legal and social**
  - **Technical**
- **Legal: society will have to accept electronic proof-of-contract**
  - **Non-repudiation more important than confidentiality**
- **Technical: overturning deeply-ingrained assumptions**





Distributed programming is different

TRADITIONAL	REVERSED
Central control	Autonomy
Common representation	Multiple representations
Closed world	Open house
Global consistency	Weak consistency
Sequential	Concurrent
Single Copy	Multiple Copies
Shared	Separate
Global naming	Context Relative naming
Complete failures	Partial Failures
Early Binding	Late Binding

The explosion of complexity

complexity	->	too many details
scale	->	too many components
federation	->	too many chiefs
heterogeneity	->	too many varieties
separation	->	too many indirections
concurrency	->	too many conflicts
reliability	->	too many faults
availability	->	too many demands
security	->	too many hackers
evolution	->	too many changes

Who's in charge here?

- A distributed system doesn't (necessarily) have someone in control
  - e.g. Internet mail
  - the world's largest functioning anarchy (27.5 million users)
- We need to add a new words to our lexicon ...

(Con)Federation

- Not the same as hierarchy
- (Many) distributed application frameworks ignore federation
  - "The interface repository and implementation repository files on all systems must contain the same interface and implementation descriptions"

How to cope with the complexity?

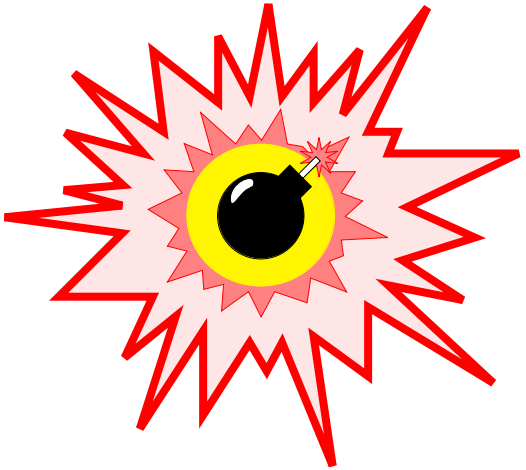
Current approaches

- **Mainly fixed on standardisation**
  - **Known as “Open Systems”**
  - **e.g. communication protocols, APIs**
- **Protocols provide interworking**
  - **Define concrete syntax and semantics of communications messages**
- **APIs provide portability**
  - **interface applications to operating systems, protocols and services**

APIs grow large & complex

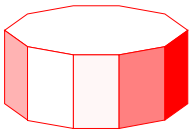
- **Rich set of concepts needed for distributed programming**
  - **Threads** **Replication** **Requests**

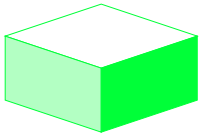
    Encapsulation **Atomicity** **(etc)**
- **API includes engineering optimisations of common combinations**
  - **fork + synchronous call + join -> asynchronous call + redeem**
- **Combinatorial explosion overwhelms the programmer**
  - **Correct usage can't be checked in advance**
  - **Compare manual for language and API**
  - **... or use of DII and IDL**

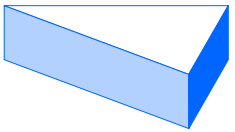


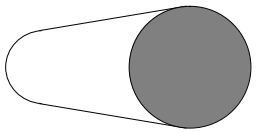
Protocols

- **Protocol standardisation necessary, but not sufficient**
  - **Protocols tailored to particular circumstances**
  - **One size will never fit all**
  - **There will always be many to choose from**
  - **Can't afford to stifle evolution**
  - **... and anyway, no-one's in a position to mandate one**
- **Conclusion: a global system will never be homogeneous**







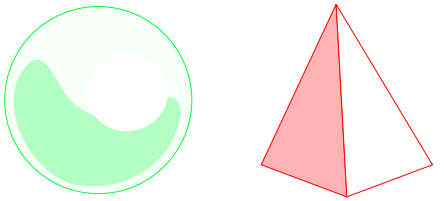


Learn from history

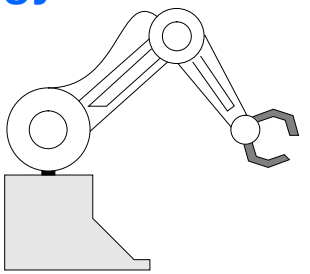
- Parallels with complexity explosions addressed in the past
  - Octal & Hex instruction sets -> assemblers
  - Assemblers -> HLLs
  - Physical -> logical -> semantic database schemas
  - 4GLs
- Always better abstractions supported by more sophisticated tools
- There is usually a rearguard action against this
  - On grounds of reduced efficiency
- But increased abstraction and automation always won
  - MIPs double and costs halve every 30 months

Abstract and automate

- **Abstractions must:**
  - hide irrelevant detail from the programmer
  - deal in independent, orthogonal concepts
  - impose the simplest possible conformance rules
  - provide the greatest possible configuration flexibility



- **Automatic tools need to:**
  - compile application programs into any suitable technology
  - optimise applications for a particular configuration
  - perform as many correctness checks as possible

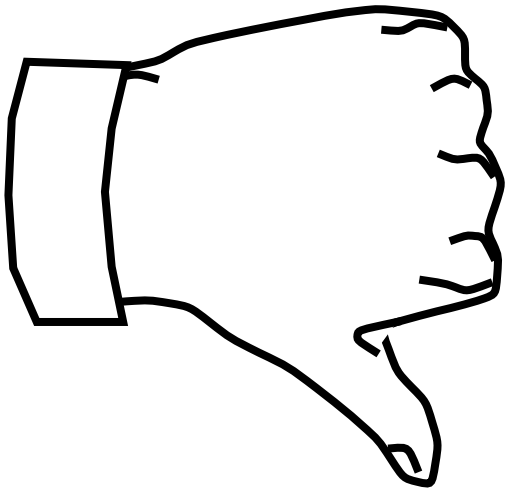


Objects: a key abstraction

- Objects hide heterogeneity inside encapsulation
- Address of remote service abstracted as object handle
  - Easily passed as invocation parameter
- Operation invocation can be automated
  - Stub compilers work from Interface Definition Language
  - Tools install different engineering optimised for different circumstances
  - ... local vs remote
  - ... transactional vs not
  - ... replicated vs singleton

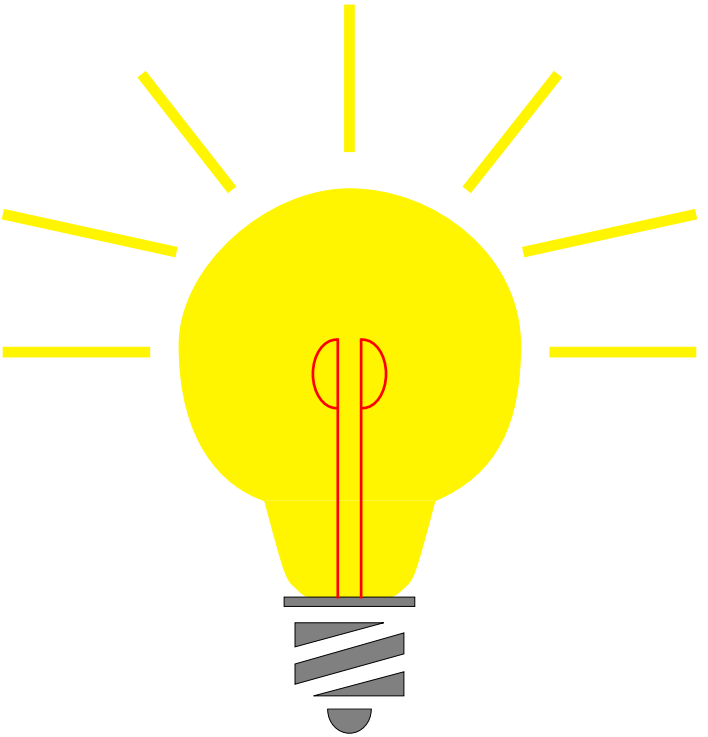
Not all objects are distributable

- **Some “traditional” features of object-orientation hinder**
  - **Class inheritance**
  - **Class variables**
  - **Encapsulation violation by other class instances or “friends”**
  - **Object models without encapsulation (e.g. CLOS, Dylan)**
  - **... all mask “hidden” communication channels**
  - **Reference equality test**



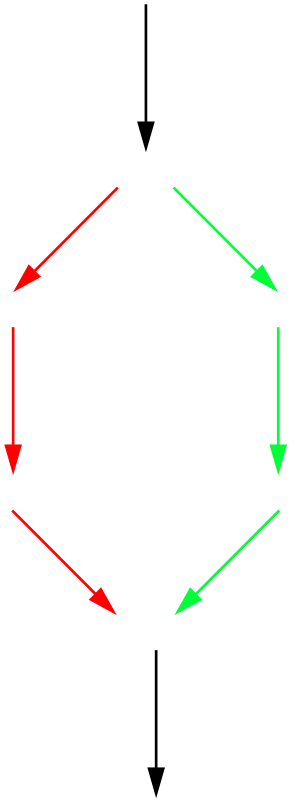
Object features to aid distribution

- **Multiple result parameters**
  - Deliver maximum possible data per invocation to beat latency
- **Multiple outcomes per operation**
  - c.f. Unix library `getchar()`
- **Multiple "interfaces" per object**
  - Encapsulated state may present many services



Concurrency abstractions

- **Distributed systems are inherently concurrent**
  - **Serialised activities must be “added back”**
- **Tree-structured activities**
  - **Orthogonal to objects**
  - **Permit nested transactions, automated transparent object replication**
  - **Asynchronous call + redeem analogous to GOTO**
- **Declarative specification of concurrency separation predicates**
  - **Tools insert appropriate engineering**



Things you can't abstract away

- **Latency**
  - Absolute maximum speed at which information can travel
  - ... and it only gets more apparent as machines get faster
  - Caching alleviates some latency, at the expense of jitter
  - Alternatively, trade off latency against consistency

- **Partial failure**
  - Distributed systems have multiple failure domains
  - Cannot be abstracted away in large systems
  - Better to expect a failure that never comes ...

Conclusion

- Increased use of large-scale distributed systems inevitable
- Federation must always be taken into account
- Standardisation is necessary, but not sufficient
- Way forward via more abstraction, better automation
- Make applications independent of their configuration, distribution engineering and support technology
- Use tools to cope with optimisation and evolution