DISTRIBUTED COMPUTING MEETS TELECOMMUNICATIONS

A J Herbert

Rapporteur: John Dobson

ANSA

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

ANSA Phase III

Distributed Computing Meets Telecommunications

Andrew Herbert

Abstract

This is a presentation to be given at the ICL / University of Newcastle Workshop on "The Future of Software", September 1995.

It focusses on the impact of deregulation and broadband technology on the telecommunications industry, leading to the uptake of object technology in initiatives such as TINA. The presentation then explores some of the extensions needed to current distributed object platforms to support interactive multi-media services.

APM.1557.01 **Approved** 25th August 1995
External Paper

Distribution:
Supersedes:
Superseded by:

ANSA

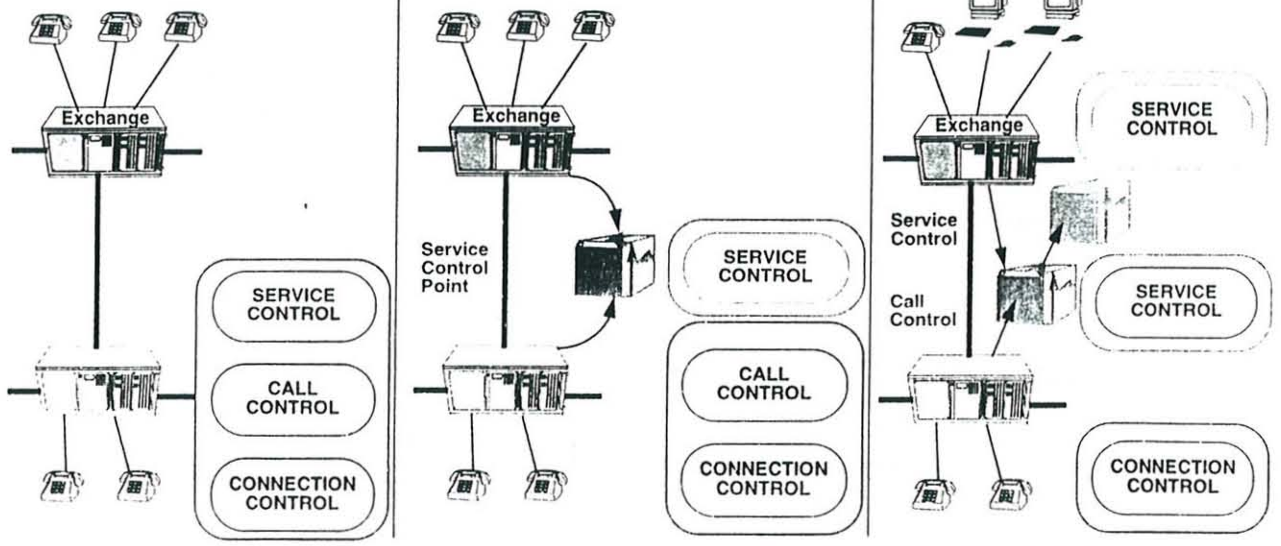
Forces for Change

- Broadband
  - optical network, ISDN, ATM
  - digital services to user
- Deregulation
  - competition
  - new players
- Interactive multi-media
  - Video-on-Demand
  - Home shopping
  - Information access
- Challenge
  - fast roll out of new services
  - exploit niche opportunities
  - supplier independence

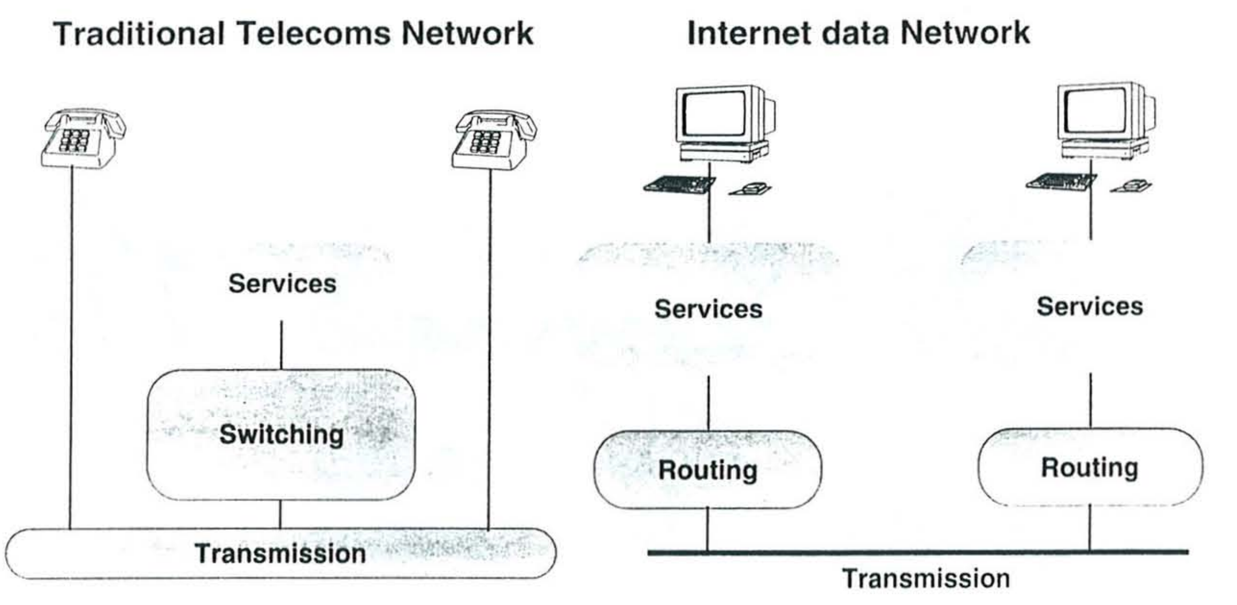
ANSA

Telecoms Architectures

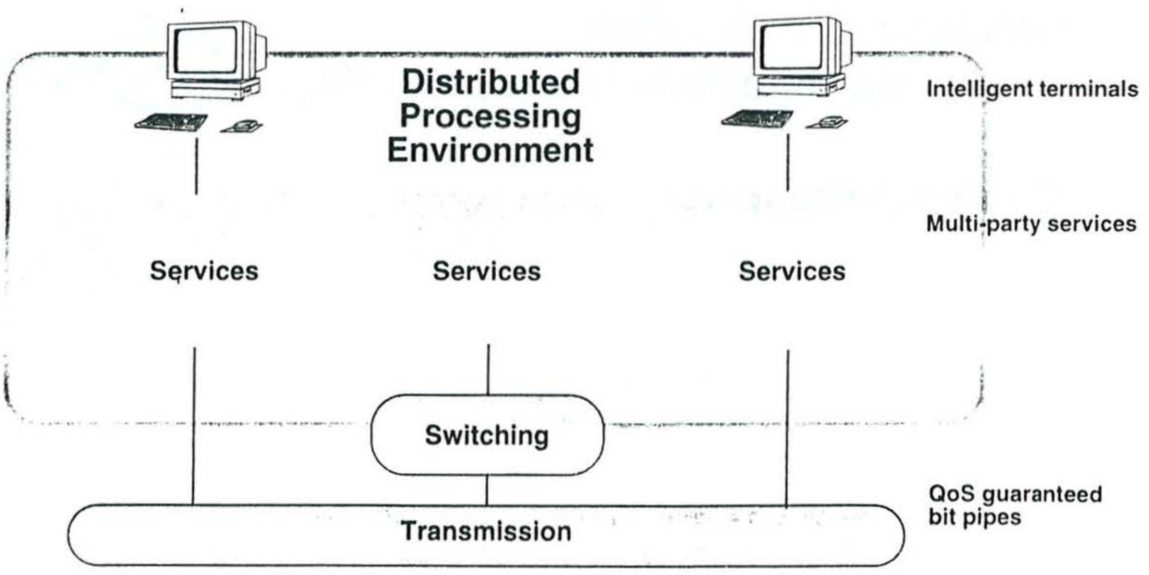
Traditional | Intelligent Network | TINA



Telecoms Network versus Internet

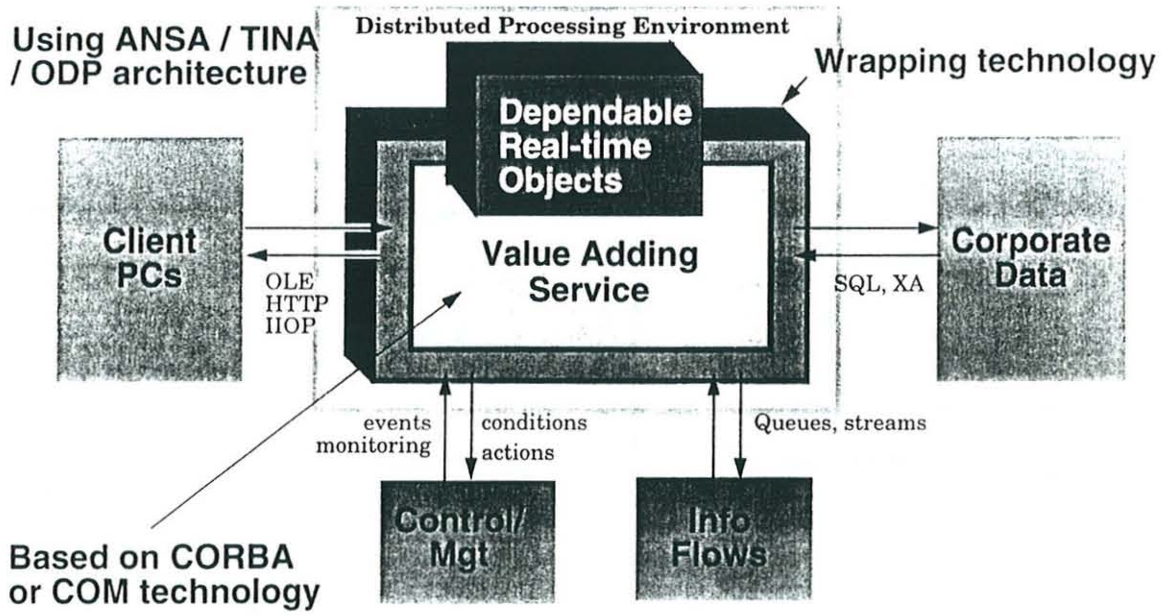


Convergence



ANSA

Distributed Processing Environment Template



ANSA

Principles for Distributed Processing Environments

Trading and Federation

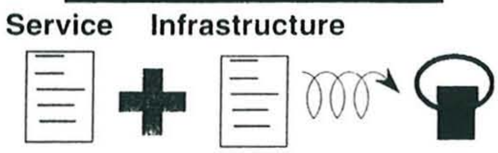
Configurable interoperability

Custom Infrastructure

One size does not fit all

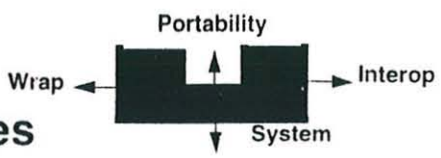
Abstract & Automate

Tools replace APIs



Modular Engineering

Architected internal interfaces



Trading and Federation

- Large systems are made up of autonomous islands (domains)
  - interconnected incrementally, no central authority
  - legacy of old technology, conflicting choices of new technology
- Administrative boundaries: where checks and accounting occur
- Technology boundaries: where protocol and data conversion occur
- Objects provides services to one another
- Object advertize services in traders
- Trader uses meta-data (type, properties) to ensure integrity
  - no suprises rule for matching allows for system evolution
- Set up *interceptors* (gateways / bridges) on demand, when trading between domains, driven by meta-data

Custom Infrastructure - Enabling Trade-offs

- Distributed systems engineering is all about trade-offs
  - ABSTRACTION versus SPECIALIZATION - the more you hide, the less control you have
  - CONSISTENCY versus AVAILABILITY - availability implies copies, increases risk of inconsistency
  - AUTONOMY versus UNIFORMITY - autonomy gives more freedom but leads to differences which increases complexity
  - SECURITY versus CONVENIENCE - security makes things harder to do
- Distributed application design versus distribution transparency
- Therefore we need a kit of *transparent* solutions and an open *nucleus* into which they slot
- Moreover the nucleus must accomodate coexistance of alternative parts for the same job

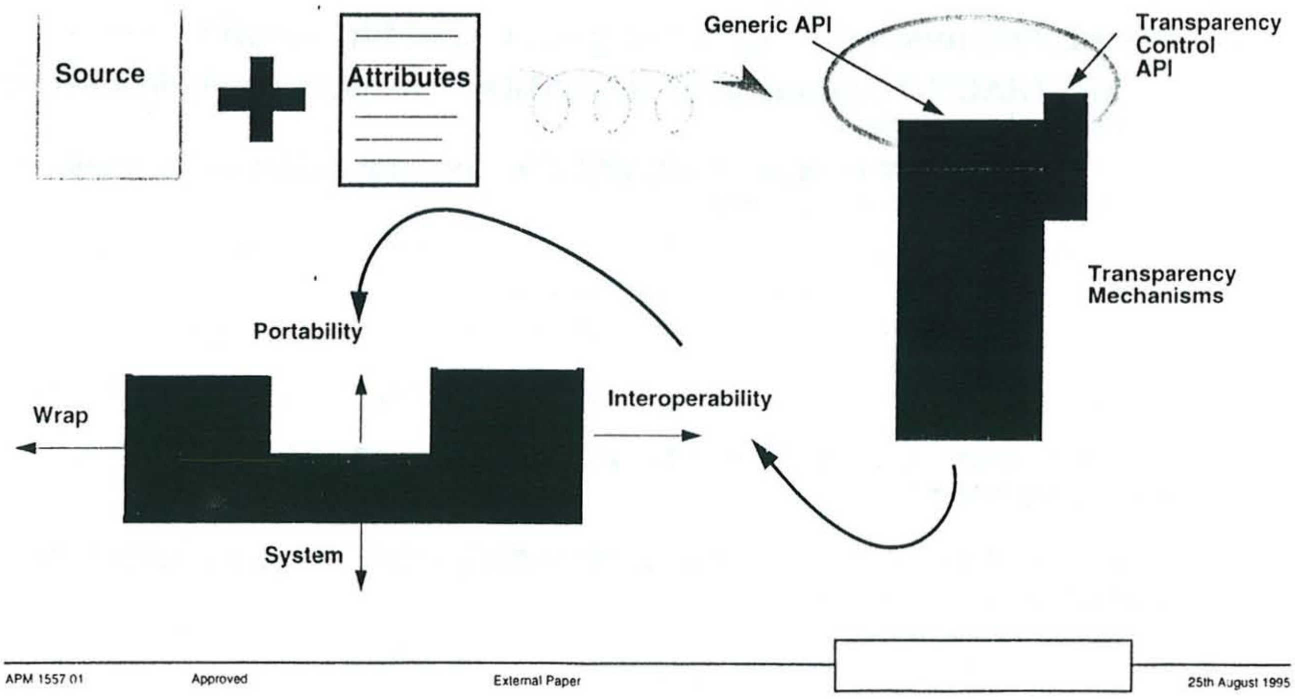
Selective Transparency

- Transparency is about hiding mechanism

- Location	don't need to know where it is to use it
- Access	don't need to know how it works to use it
- Migration	it can move while you're using it to balance loads or reduce latency
- Replication	there may be copies for reliability and/or availability
- Persistence	it only gets resources when it needs them
- Partial Failure	it always gets to a consistent state
- Federation	you don't have to have the same administrator to use it

- Selective transparency requires
  - same API for core functions across all transparencies
  - extra management functions for controlling each transparency

Abstract + Automate + Modular Engineering



ANSA

Programming for Distributed processing Environments

- Rich set of concepts needed
  - threads for concurrency
  - requests and replies (symmetrical c.f. procedure calls)
  - replication for availability, fault tolerance
  - atomicity for failure recovery, concurrency control
  - ......
- Optimized engineering for common cases
  - e.g. forked call -> asynchronous call to save a local thread
- Special engineering for special cases
  - e.g. spawned atomic request -> start new top level transaction
- Combinatorial explosion in functions overwhelms the programmer

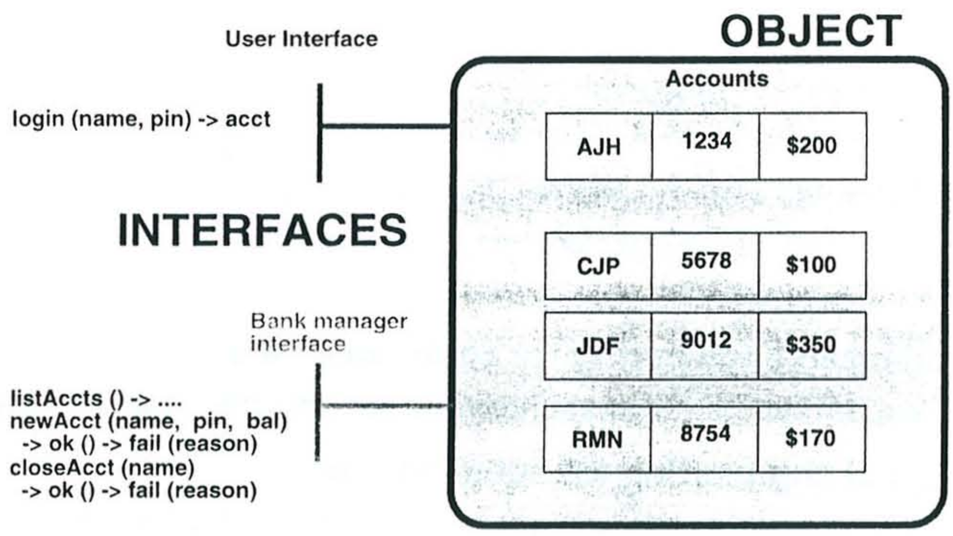
ANSA

Let Compilers and Tools Take The Strain

- Exploit abstraction, program in application oriented concepts
  - most aspects of OO really help, some hinder
- Simple (pre-processor) extensions go a long way
  - especially if leveraging an OO language
- orthogonality - e.g. "dot" and "bar" *vs.* threads and RPC API
  - languages minimize complexity without losing scope for optimization
- declarative - state requirements and policies not mechanisms
  - point already proven by IDLs and stub generators
  - decouple applications from engineering - ANSA PREPC experience
- strong type checking for safety and confidence

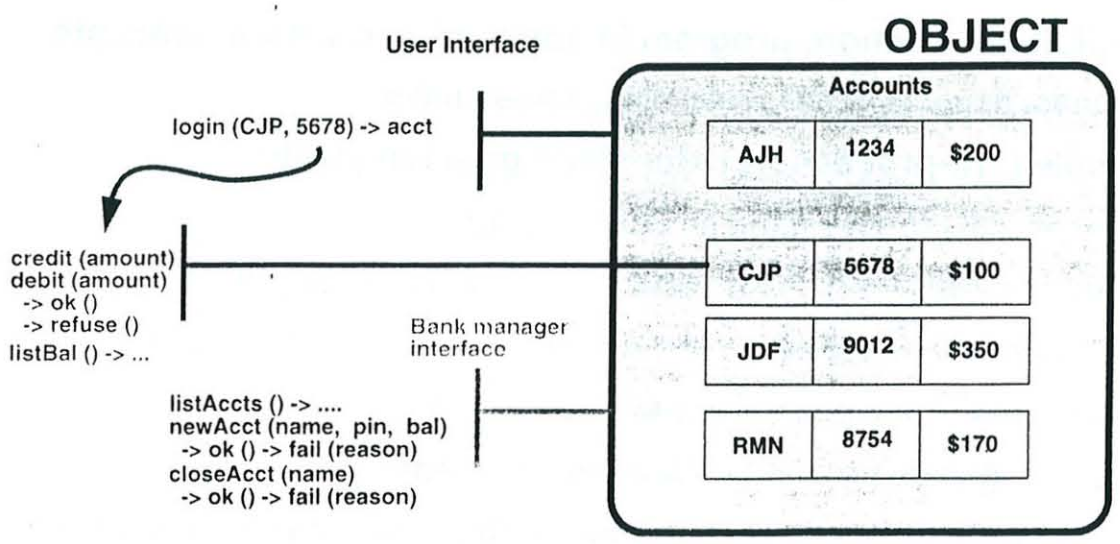
ANSA

ODP / TINA / CORBA Distributed Object Model (1)



ANSA

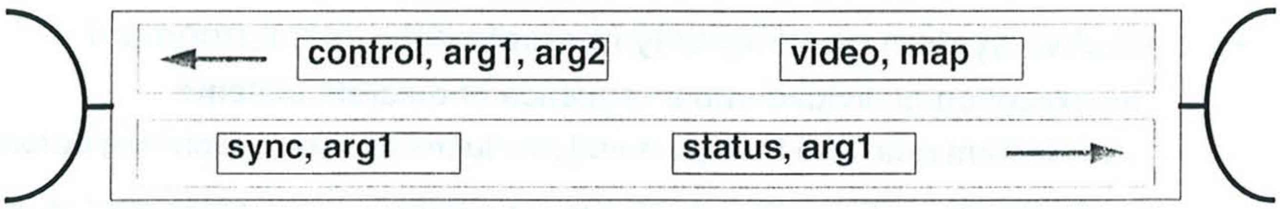
ODP / TINA / CORBA Distributed Object Model (2)



SpecificTelecoms Requirements

- Extensibility, scaleability, federation, dependability
- *Support for multi-party interactions*
  - explicit binding
- *End-to-end quality of service control)*
  - explicit binding
  - fine-grained resource management
- *Routing audio, video etc to applications*
- *Predictable computation*
  - synchronous programming
- *Performance*
  - DPE close to bare hardware, perhaps merged with microkernel?

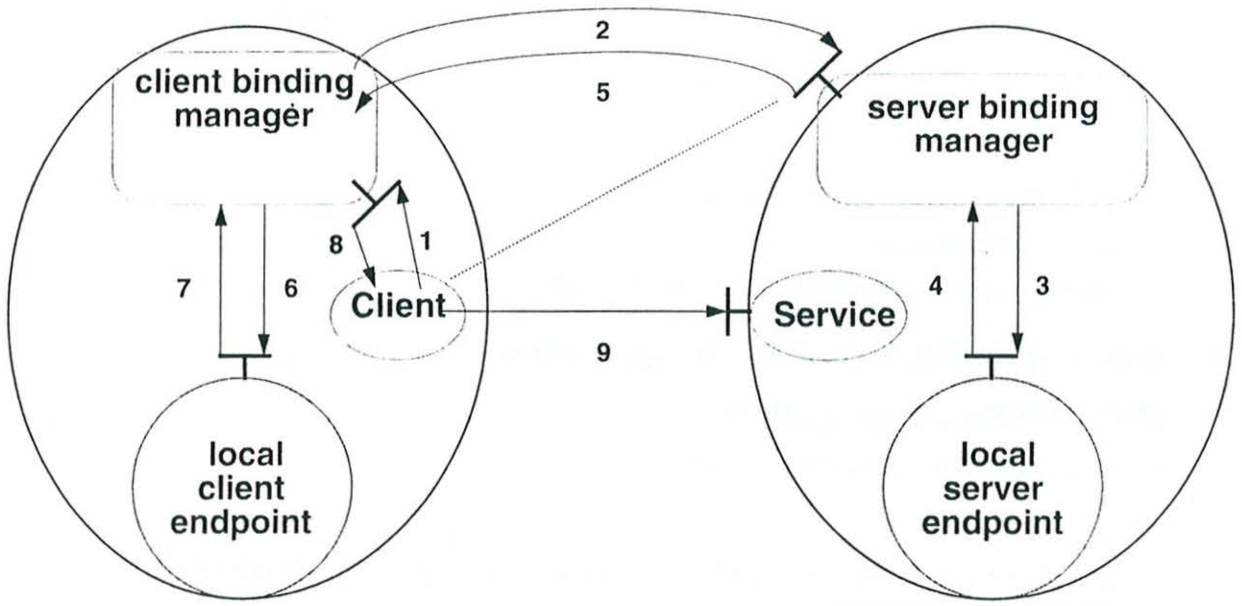
Streams



- *A stream must be bound at both ends before it can be used*
- *A stream has a set of flows*
- *A flow has a set of frames (or signals) and a direction*
- *A frame has a name and a set of typed arguments*
- *Streams are typed and can be conformance type checked*
- *Frames transmitted by non-blocking writes, read by blocking reads*

ANSA

Binding - Another Distributed Application



ANSA

Synchronous Programming

- a reactive system continuously interacts with its environment
  - its execution is divided into a sequence of discrete instants
  - each instant reacts to its inputs and produces the corresponding outputs
- the synchronous hypothesis states all reactions are instantaneous
  - simplifies reasoning by removing all concurrency between instants
  - execution of communicating threads in the same instant are serialised
- deterministic behaviour
  - bounded execution paths, calculable in advance
  - with guaranteed resources:
  - programs have predictable timing and reproducible behaviour [even in asynchronous systems]

Example Synchronous Language

```
<exp>           = <exp> | <exp> ; <exp> | <exp> || <exp>
signal          = signalName [attributelist]
                            "(" { typeExpression } ")"
direction       = ">>" | "<<"
flow            = direction  [attributeList] "(" { signal }
")"
stream          = "stream" [attributeList] "(" {flow} ")"
transmission    = unit "!" signalName block
reception       = unit "?" signalName
condition       = "(" {reception} ")"
await           = "await" condition
watchdog        = "during" block "watch" condition block
presence        = "present" condition
```

Key Points

- Distributed Objects
- Blurring of Computing / telecoms distinction
- Modular Infrastructure
- More of the OS becomes an "application"
- Computing with Time and Quality of Service
- Custom hardware turning into software on commodity hardware

DISCUSSION

Rapporteur: John Dobson

Lecture One

Professor Martin asked whether Dr Herbert's use of the word "service" sometimes confused the telecommunications community, who had their own interpretation of the word.

Dr Herbert replied that this did happen on occasion, but the confusion sometimes served to clarify what the word means in the telecommunications community and the distributed systems community. Professor Katzenelson asked whether the approach outlined was adequate to deal with real-time problems.

Dr Herbert replied that an efficient operating system (in performance terms) could be engineered by placing the distribution mechanisms in the operating system closer to the kernel, but additional abstractions were needed to handle real time, particularly in the areas of control over multiplexing and time allocation to resources. Professor Randell asked about the link between objects considered as bits of mechanism and objects considered as linguistic constructs. Dr Herbert replied that encapsulation (i.e. the ability to move an object about as a single entity) and genericity (types of object subject to a common management regime) were what were important, and these could be seen mechanistically or linguistically. Also objects might require different access paths under different kinds of transparency attribute. These properties could not be handled if objects were seen as possessing methods. Professor Randell further enquired if current work on evolving objects was relevant to these points. Dr Herbert replied that it was. Professor Kopetz asked whether encapsulation should include temporal properties. Dr Herbert replied that it should, but currently no system supported that.