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**

Training

ANSAwise - Replication in Distributed Systems

Chris Mayers

Abstract

Organizations require dependable services.

Dependability is a complex problem, and many techniques need to be brought together to achieve it. Replication techniques are part of the solution.

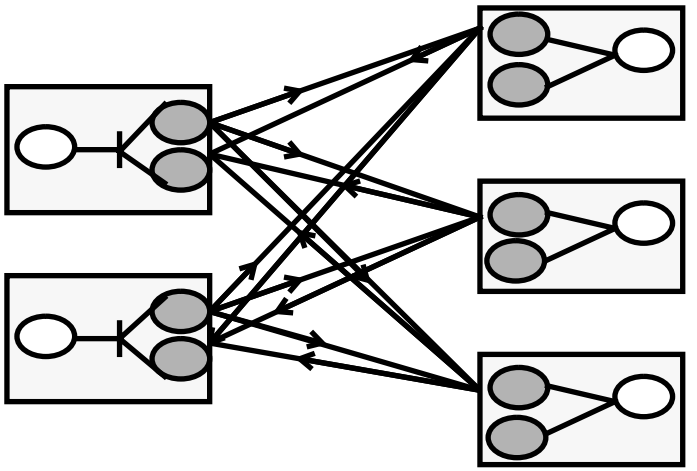
This module of the ANSAwise training programme discusses the key concepts of group communications and the issues that arise from them (in particular, ordering). It then outlines the likely future of replication in CORBA products.

[This module is derived from APM.1358; it omits the background on ANSAware designs for group communications (GEX).]

APM.1743.02 **Approved** 4th April 1996
Briefing Note

Distribution:
Supersedes:
Superseded by:

Replication in Distributed Systems



In this session

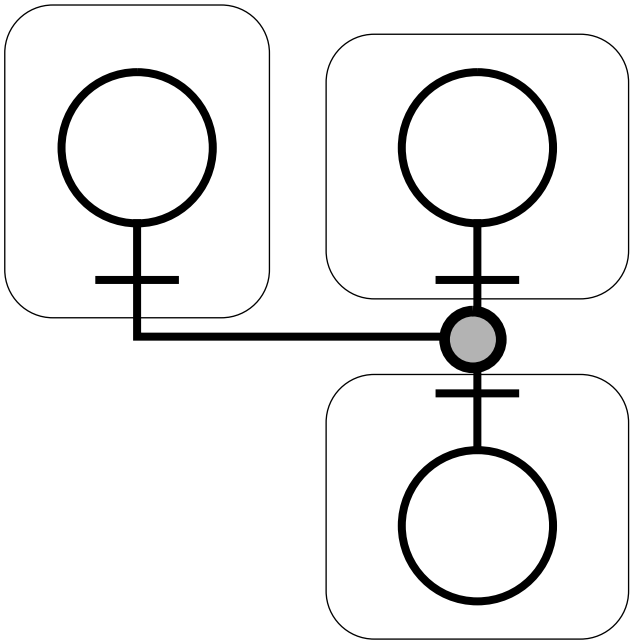
- Explain how group communications can achieve dependable delivery
  - to replicated implementations
- Explain how interface groups implement replication transparency
- Explain the protocols used to support interface groups
- Suggest how replication will evolve in CORBA

The purpose of groups

- **Interface groups are a construct that help you build dependable interfaces**
- **To be dependable, objects must be reliable (fault-tolerant). Reliability can be achieved by replication**
  - **keeping at least two copies; this is called *redundancy***
- **Redundancy can also improve performance**

Replication transparency

- **Replication Transparency**
  - **application need not know how many copies**



  - **application only sees a single interface**

Replication and Trading

- **Replication is appropriate when you want to keep N copies of the same information in step**
  - **for example, for dependability**
- **Trading is appropriate when you want to choose 1 from N copies of the same service**
  - **for example, for load-sharing**
- **If a service fails, a client can also use a trader to find another copy of the same service (a different object)**
  - **this can improve availability, but does not give access to the *same information*...**
  - **... unless information is kept in step by some other means**

Design issues for group communication

- **Addressing**
  - address list, group address, source address, or functional address
- **Reliability**
  - reliable or unreliable
- **Ordering**
  - none, FIFO, causal, total

More design issues for group communication

- **Delivery semantics**
  - **k-delivery, quorum(majority) delivery, or atomic delivery**
- **Response semantics**
  - **none, one, many, or all**
- **Group structure**
  - **closed or open, static or dynamic**

Reliable Broadcast

- **The minimum useful group communication guarantees**
  - ***Validity*: a correct message is sent to all replicas**
  - ***Agreement*: all processes agree on the set of messages delivered**
  - ***Integrity*: no spurious messages are ever delivered**
- **...this is known as Reliable Broadcast**

Reliable Broadcast Guarantees

- This must be guaranteed even if, during the broadcast
  - the client crashes
  - one or more of the group members crashes
  - new members join the group
  - existing members leave the group
  - crashed members rejoin the group
  - ... and any combination of the above, possibly more than once
- This sounds difficult enough to guarantee
  - perhaps surprisingly, it may be insufficient

The ordering problem

- **Reliable Broadcast guarantees that all messages sent will be received**
  - **but not in *which* order they will be received**
- **This is a familiar problem with postings to bulletin boards**

	From	Subject
20	G. Joseph	Microkernels
21	A. Hanlon	Mach
22	A. Sahiner	Re: RPC performance
23	M. Walker	Re: Mach
24	T. L'Heureux	RPC performance
25	A. Hanlon	Re: Microkernels
...	...	...

Ordering requirements

- **Ordering failures arise because of**
  - **partial broadcasts (to some replicas, but not others)**
  - **... this is prevented by Reliable Broadcast**
  - **variable propagation delays**
  - **... this isn't**
- **Applications may require two independent properties**
  - **total ordering (all replicas see the same order)**
  - **causal ordering (all replicas see a 'right' order)**

Causal But Not Total Order - A

- **Site A sees this order...**

	From	Subject
20	G. Joseph	Microkernels
21	A. Hanlon	Mach
22	M. Walker	Re: Mach
23	T. L'Heureux	RPC performance
24	A. Sahiner	Re: RPC performance
25	A. Hanlon	Re: Microkernels
...	...	...

Causal But Not Total Order - B

- ... site B this order

	From	Subject
20	G. Joseph	Microkernels
21	A. Hanlon	Mach
22	A. Hanlon	Re: Microkernels
23	M. Walker	Re: Mach
24	T. L'Heureux	RPC performance
25	A. Sahiner	Re: RPC performance
...	...	...

- A and B are both 'right', but different

Total But Not Causal Order

- **Every site (replica) sees the same ‘wrong’ order**

	From	Subject
20	G. Joseph	Microkernels
21	A. Hanlon	Mach
22	A. Sahiner	Re: RPC performance
23	M. Walker	Re: Mach
24	T. L’Heureux	RPC performance
25	A. Hanlon	Re: Microkernels
...	...	...

Total Order is not always required

- Total ordering may be inappropriate for some applications
  - it might be useful to read “Re: RPC performance”, even if “RPC performance” is unavailable...
  - ... and later postings can supersede earlier ones anyway
- Total ordering is often impractical
  - a single point may be required to allocate sequence numbers...
  - ...and a bottleneck

The good news

- If (synchronous) RPC is used, causal ordering comes for free
- Note that in a multithreaded client there is no causal ordering *between* threads
  - if needed it is the client's responsibility to enforce such order...
  - ... by the usual synchronization mechanisms

The bad news

- You've probably guessed that causal+total ordering is inefficient
  - ... and, alas, you are right...
  - ... causal (but not total) ordering tends to be more efficient
- So, distributed programming systems can offer the application designer a choice
  - ... and you must understand the performance implications of each choice
  - ... and whether the choice is appropriate for your application

Worse news

- You may need to design different ordering requirements for different operations
- For example, with the bulletin board
  - causal order: for posting messages
  - causal+total order: for adding a new subscriber
  - ... to avoid a new subscriber being added twice by different (uncoordinated) sites
- Yet another even stronger ordering (sync-ordering) is needed to resolve interactions
  - ... when some operations have causal order, and others total order
  - sync order: for deleting a subscriber
  - ... so they can't read messages from other sites afterwards

Even worse news

- That was just the ordering problem
  - you may have to juggle choices in all the other design issues as well
  - ... delivery semantics, response semantics, group structure,...
- Different group communication products offer different combinations of choices

The positive side

- The application designer does have a choice
  - causal+total order is nearly always right...
  - ...others can be regarded as an application-specific optimization
- A group communications product will be better than any ad-hoc solution
  - better proven
  - easier to use
  - more effective

ISIS

- A distributed programming environment for group communications
  - no IDL
- Originally developed at Cornell University in 1983
  - now owned by Stratus
- Widely deployed in financial trading applications, supporting
  - fast response
  - high reliability (fault tolerant)
  - coordinated information

Ordering in ISIS

- **Supports all orderings**
  - **unordered (FBCAST)**
  - **causally ordered (CBCAST)**
  - **totally ordered (ABCAST) - default**
  - **sync ordered (GBCAST)**
- **Efficient implementations**
  - **will use network multicast if available**
  - **careful use of piggy-backed requests and responses**

ISIS group types

- **Peer groups**
  - all messages go to all members of the group, including the sender
- **Server groups**
  - messages go to all members of the group
  - ... one responds, determined by the group itself
- **Client-server groups**
  - one member processes the request...
  - ...response is sent to the client, and all the other members of the group
- **Subscription groups**
  - for one-way messages

ISIS group membership

- Groups can be created at any time
- Processes can join and leave groups...
  - ...and processes can belong to more than one group at the same time
- ISIS monitors the members of the group
  - to see if they are still alive

ISIS group state transfer

- When joining a group, the new member can request the ‘application state’
  - retrieved from an existing member
- ISIS guarantees that the application state is consistent with the other members
  - even though members join, leave, and crash
- ... the key to distributed failure recovery

Architectural support for groups

- Groups should be transparent to applications
- Groups should be definable in IDL
  - as groups of interfaces of the same interface type
- Groups need to be configured flexibly
  - with mechanisms and policies to tailor them
- Groups must manage themselves
  - external managers enforce the client policy (for instance, the minimum number of members of a group)

Group configurations

- Groups can use *active* or *passive* replication
  - active replication to reduce recovery time
  - passive replication to reduce overheads
- In principle, group interfaces are the general case
  - every interface should be a group interface!
  - ...singletons are the special case of a group with 1 member
  - ...singletons are an optimization

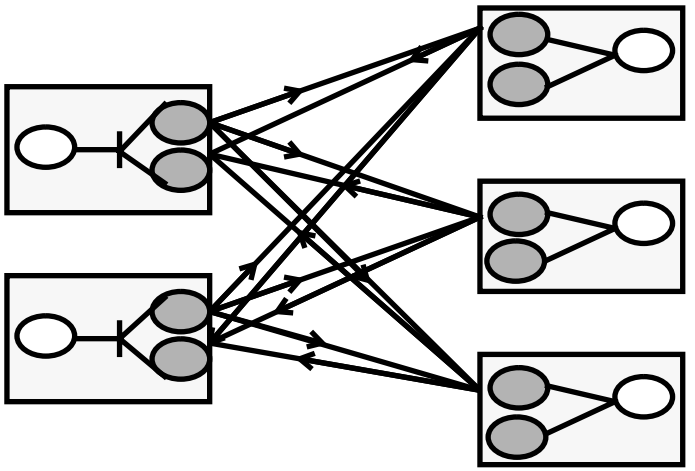
Transparency of groups

- The use of an interface group is transparent to the client
  - the difference is largely handled by the infrastructure
- The use of an interface group is not quite so transparent to the members
  - because of the need for application state transfer...
  - ... but application state transfer may be needed for persistence and migration anyway

Replication and CORBA

- **Iona: Orbix+ISIS**
- **Chorus: COOL/ORB**
- **Teknekron: Object Bus**
- **Electra**

The Group-Group N:M problem



- **Need to reduce the number of messages**
  - **use a multicast protocol**

Some experimental results

- Replication transparency really works
- Full non-selective replication transparency is possible
  - we have made it work...
  - ...but it was difficult to achieve
- Group management and configuration are central problems
- Insertion of user mechanisms and policies requires much work
- Performance is disappointing without multicast protocols

Implications for dependability in general

- **Transparent solutions will often be unacceptable**
  - **unacceptable performance**
  - **unacceptable cost**
- **The application must participate in dependability provision**
  - **to exploit semantic knowledge**
  - **to relax synchronization requirements**
  - **to set the configuration, protocol parameters, synchronization mechanisms and policies, and state synchronization**
- **The infrastructure should provide a framework for constructing dependable applications**
  - **use standard components...**
  - **... or provide your own**

Summary

- **Replication transparency uses special mechanisms to make sure the group members are consistent**
  - **for instance, it may use multi-point channels and special protocols**
- **Implementing replication transparency efficiently is difficult**
  - **it may need information from the application**
  - **it is under active research in the distributed systems community**
- **Effective group communications products are available**
  - **but need to be assessed carefully against your needs**
- **Group communication gives replication of messages**
  - **but for dependability, reliable, persistent storage is also needed**

More information?

- **For more information**
  - **on ISIS, see *Distributed Systems Concepts and Design* by Coulouris, Dollimore and Kindberg (Addison-Wesley)**
  - **on ANSA groups, see *A Model for Interface Groups* (AR.002.01)**
- **For a comparison of ISIS, a CORBA product, and ANSAware**
  - **see *Distributed Systems Engineering Vol 1 Number 4 (June 1994)***
  - **note that the version of ANSAware (3.0) discussed did not have any group support**