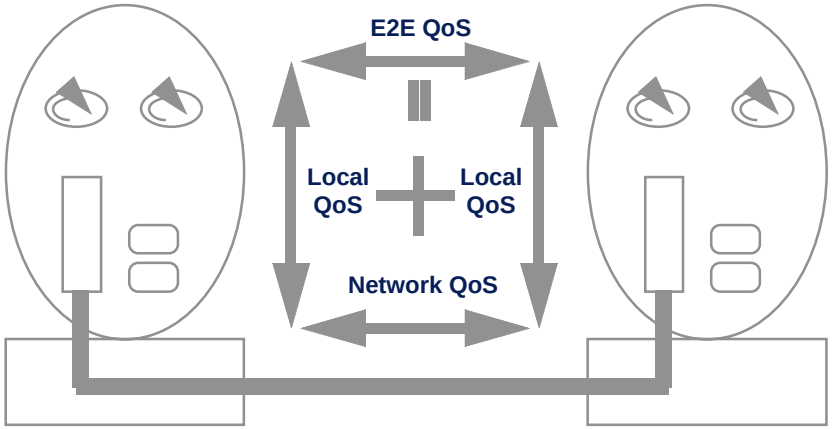
DIMMA Progress Report



Ian Macmillan

ian.macmillan@ansa.co.uk

Goals

- Microkernel (component based) ORB for applications
- ... requiring “soft” real-time QoS and MM flows
- Control over resource sharing
  - explicit binding allowing QoS to be specified
  - ... plus supporting engineering mechanisms
- Flexibility
  - Support a wide range of QoS policies
  - Lightweight <-> full function instantiations
  - Simple to add new protocols
  - Support a variety of programming “personalities”

Results

- DIMMA 1.0 (Nov 96)
  - microkernel ORB with MM extensions
  - JET (CORBA) & ODP programming personalities
    - incorporating flow interfaces
  - application multi-tasking
  - IIOP & ANSA Flow protocol (multicast UDP)
- DIMMA 1.1 (Apr 97)
  - Configurable tracing
  - Improved JET <-> CORBA compliance
  - Restructured source and build system

DIMMA 2.0

Available now

- QoS controlled Explict Binding
- Populated resource control framework
- Revised protocol framework & reworked
  - IIOP with resource control
  - ANSA Flow protocol with resource control
- Dynamic protocol loading
- Improved robustness and implementation

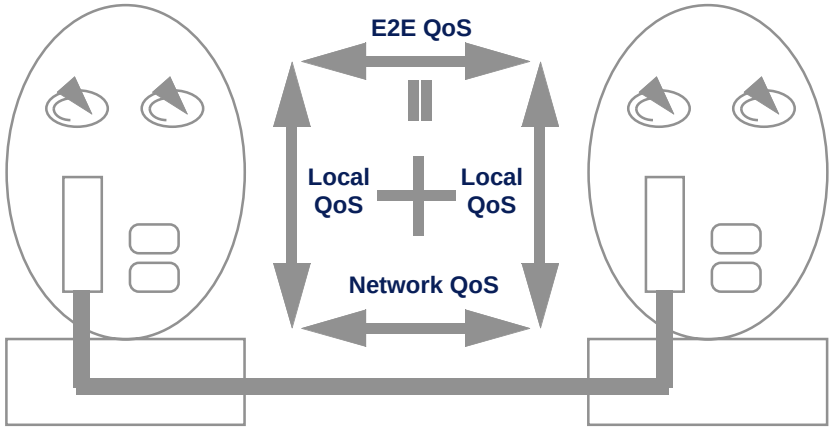
Benefits over Commercial ORBs

- Highly modular, flexible architecture
- Lightweight <-> full function instantiations
- Multiple programming personalities
- Advanced protocol framework
  - enables reuse of layered modules
- Dynamic protocol loading
  - easy to add new protocols
- Provides both MM and RT functionality
  - flow interfaces (with range of service qualities)
  - resource control framework & Explicit Binding

Relevance

- Input to future telecom ORBS (ReTINA)
- Prototype ATM connection mgmt architecture
  - exploring use of resource aware u-kernel (DCAN)
- Currently in discussion with commercial ORB vendors regarding use of DIMMA technology
- OMG already taken up flow ideas & explicit binding
  - see responses to Streams RFP (issued Aug 96)
- Influence OMG real-time CORBA RFP (Dec 96)
  - not directly but via vendors and projects
- Technology delivered to you as source code

DIMMA



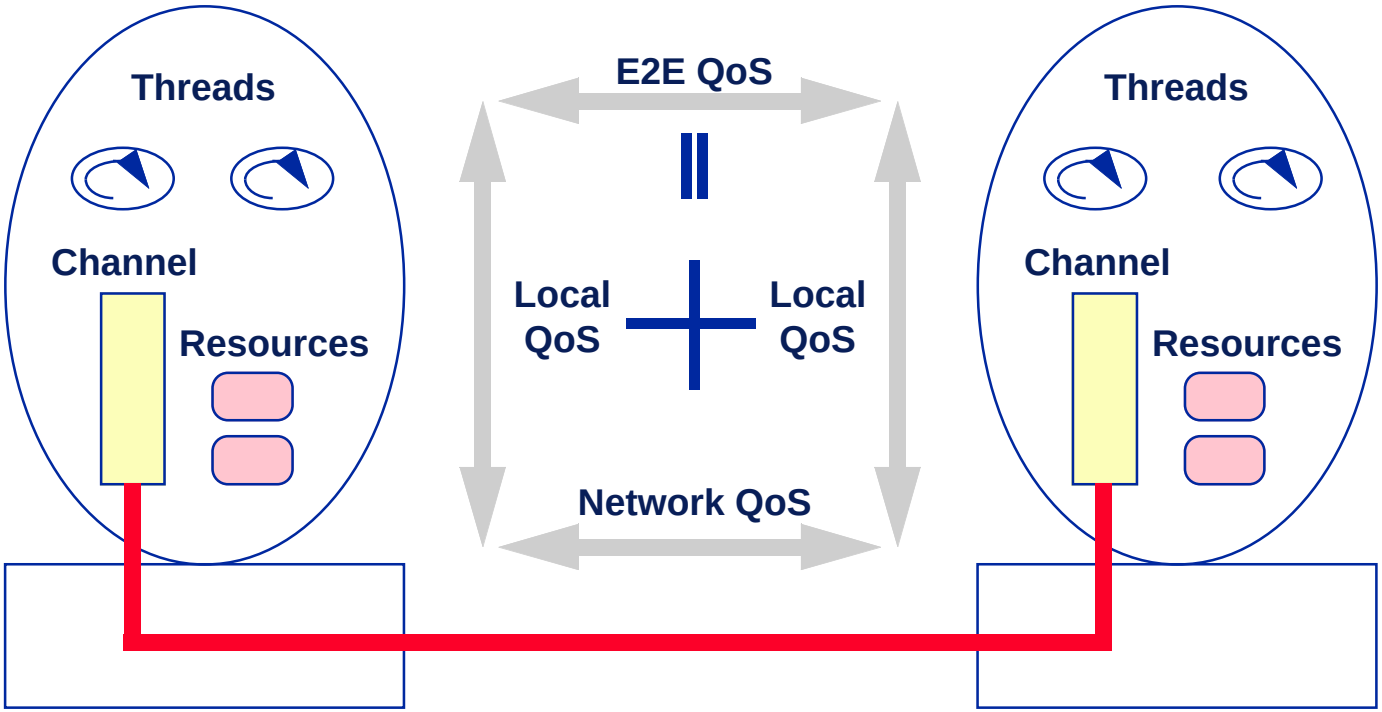
Resourcing the ORB

Why Resource Control?

- Real-time and multi-media need specific QoS
  - required end to end at the application
  - must be maintained during varying load
- Implies resources available when needed
- But resources often scarce and hence shared

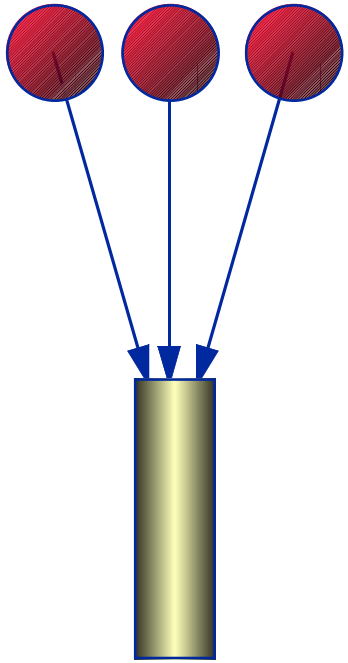
=> Sharing must be controlled

Which Resources?

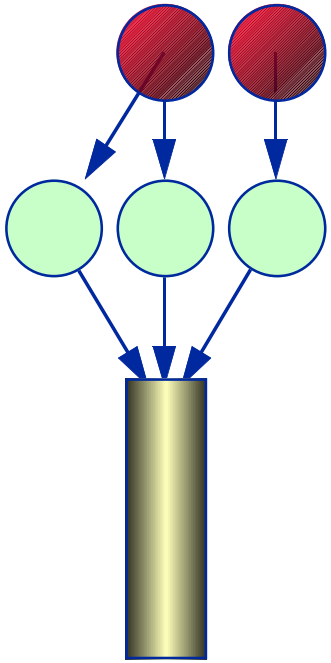


Channel Multiplexing

Objects sharing channel



Plus multiple Invocations

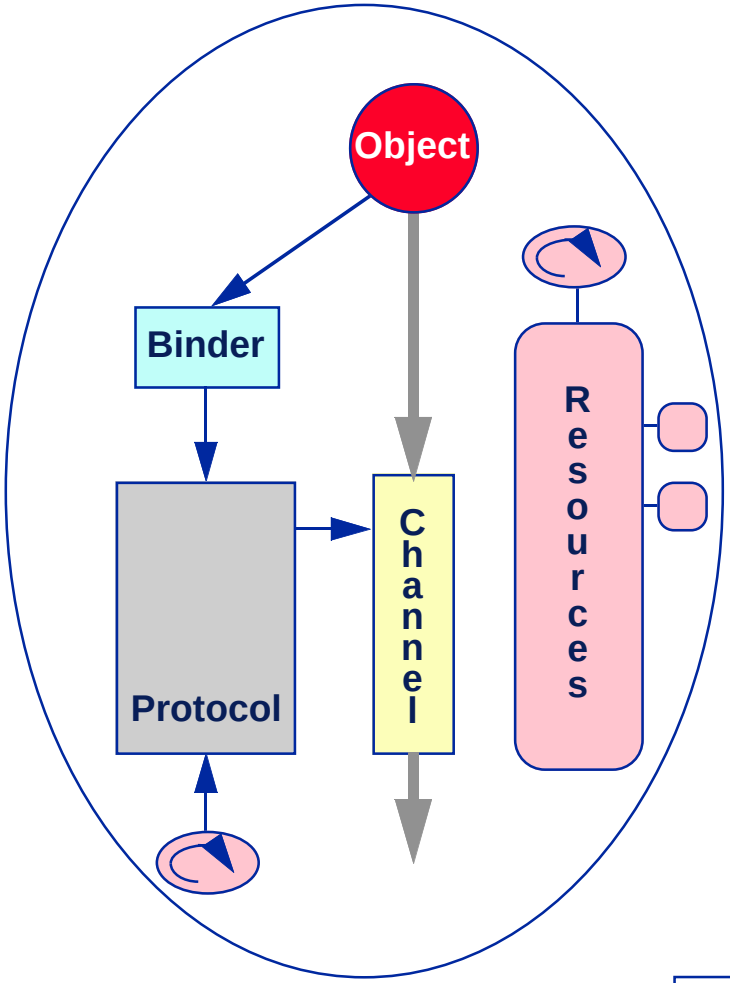


CPU Multiplexing

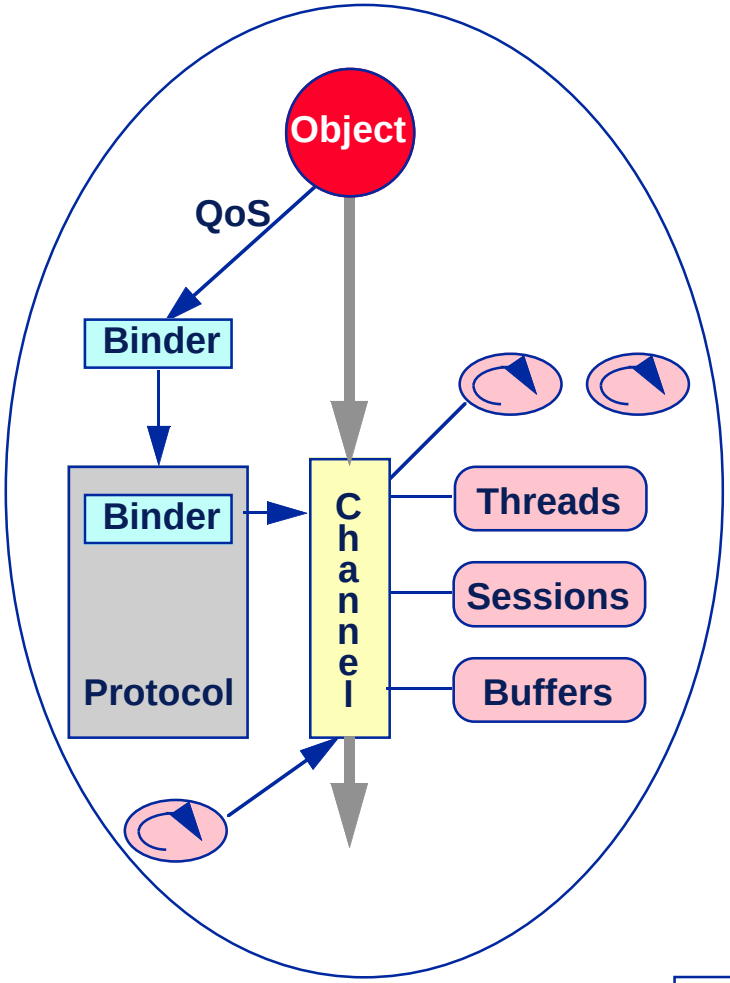
Invocation concurrency

Shared read/demux Thread

Vanilla ORB Capsule



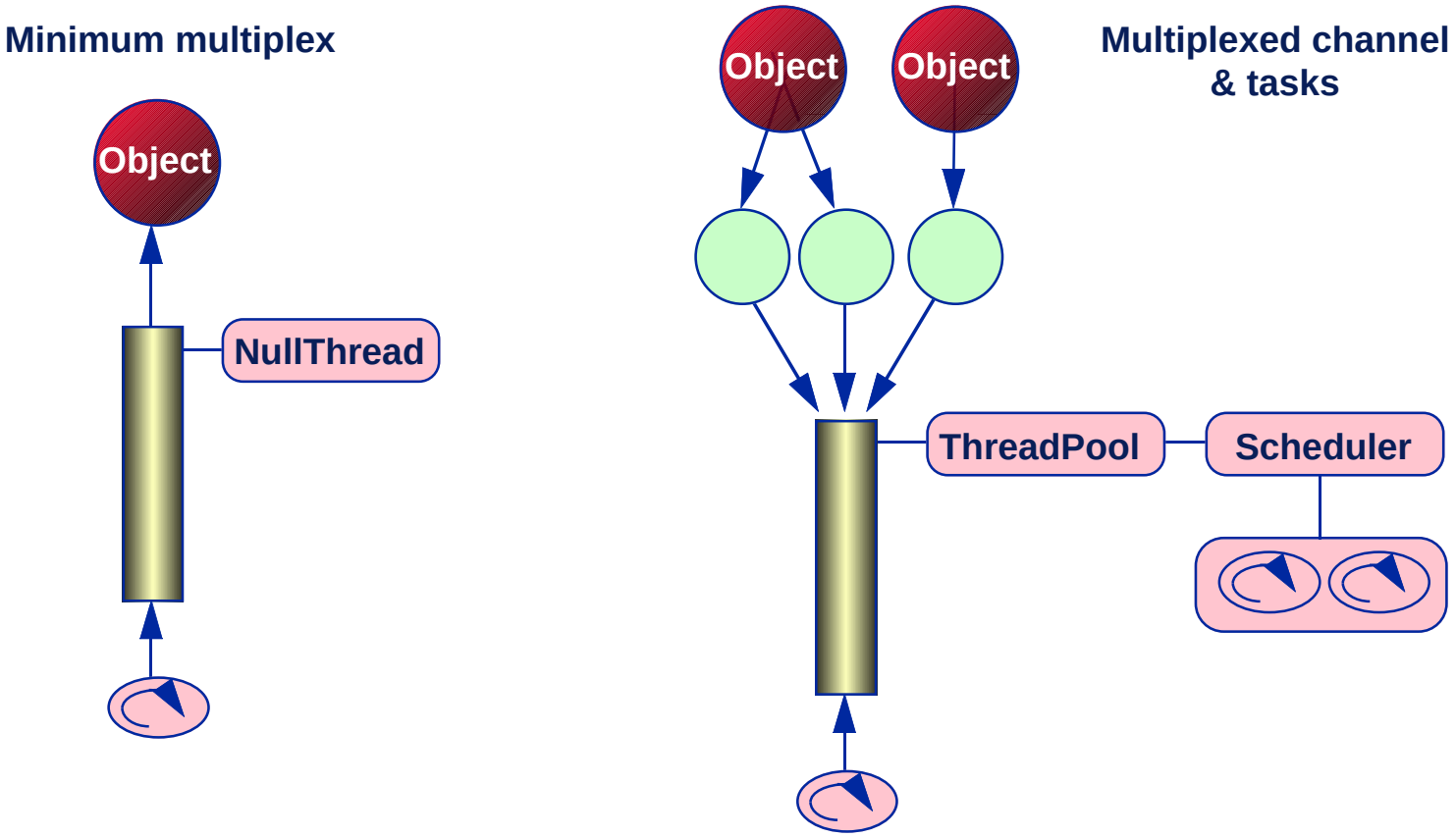
Resourcing the Channel



Components

- Resource control architecture
  - Factories
  - Pools
- Threading framework
  - Null
  - Task
  - Thread
- May be configured and composed
- ... to support diverse range of policies

Example Configurations



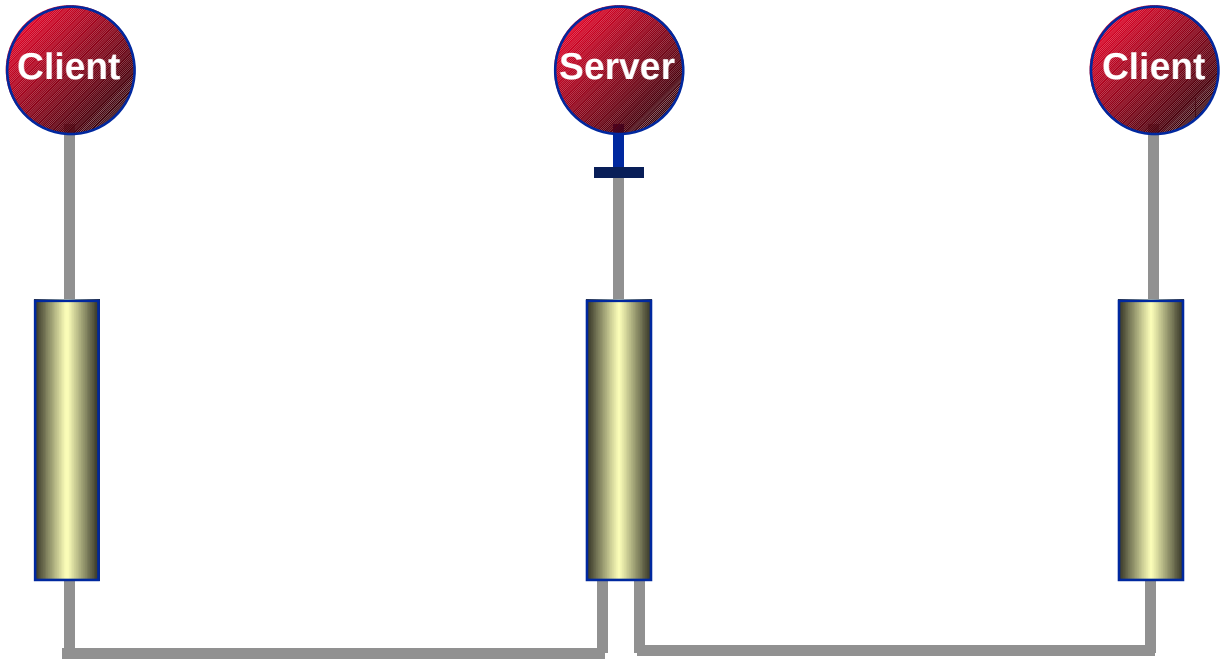
Implementation Pitfalls

- Failure to observe strict separation of concerns
  - allowing mechanism to determine policy
  - functional overlap preventing fine grain control
- Implicit assumptions
  - e.g. memory management policy
- Regarding all resources as equal
  - e.g. active tasks are not like passive buffers

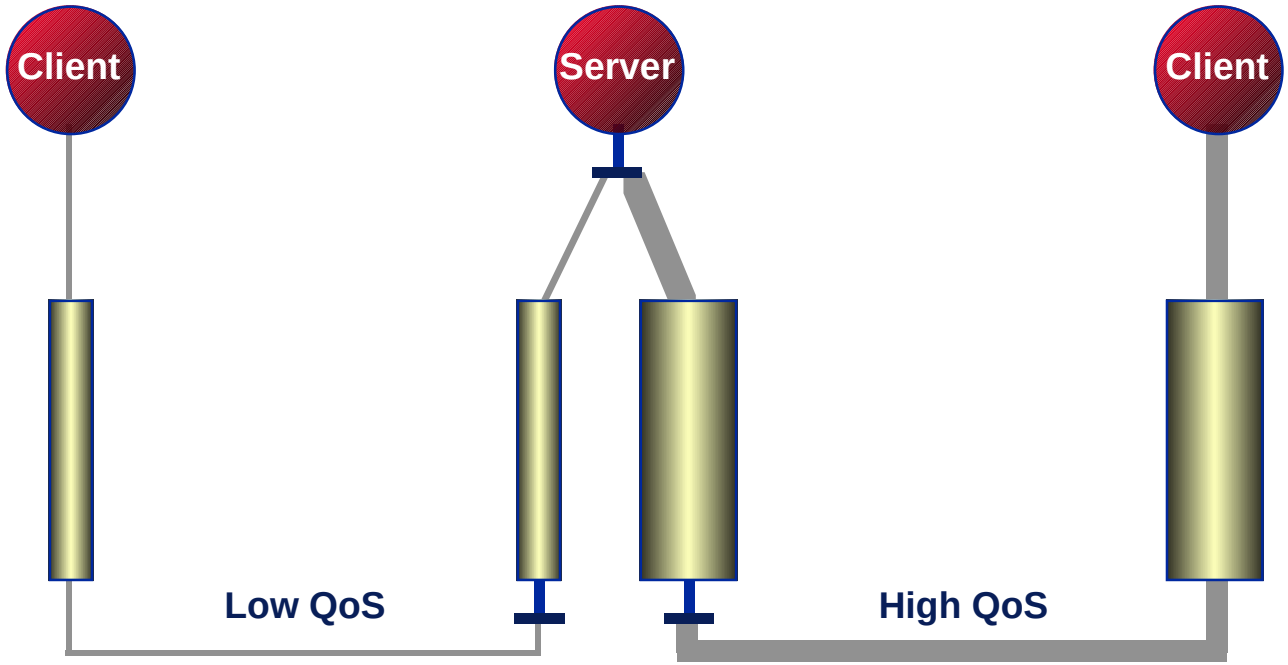
Specifying QoS

- QoS is required between application endpoints
- ... on a per connection basis
  - e.g. different client instances using the same server interface may desire different QoS.
- QoS is determined at bind time
- ... requiring additional binding apparatus
  - ... taking QoS attributes

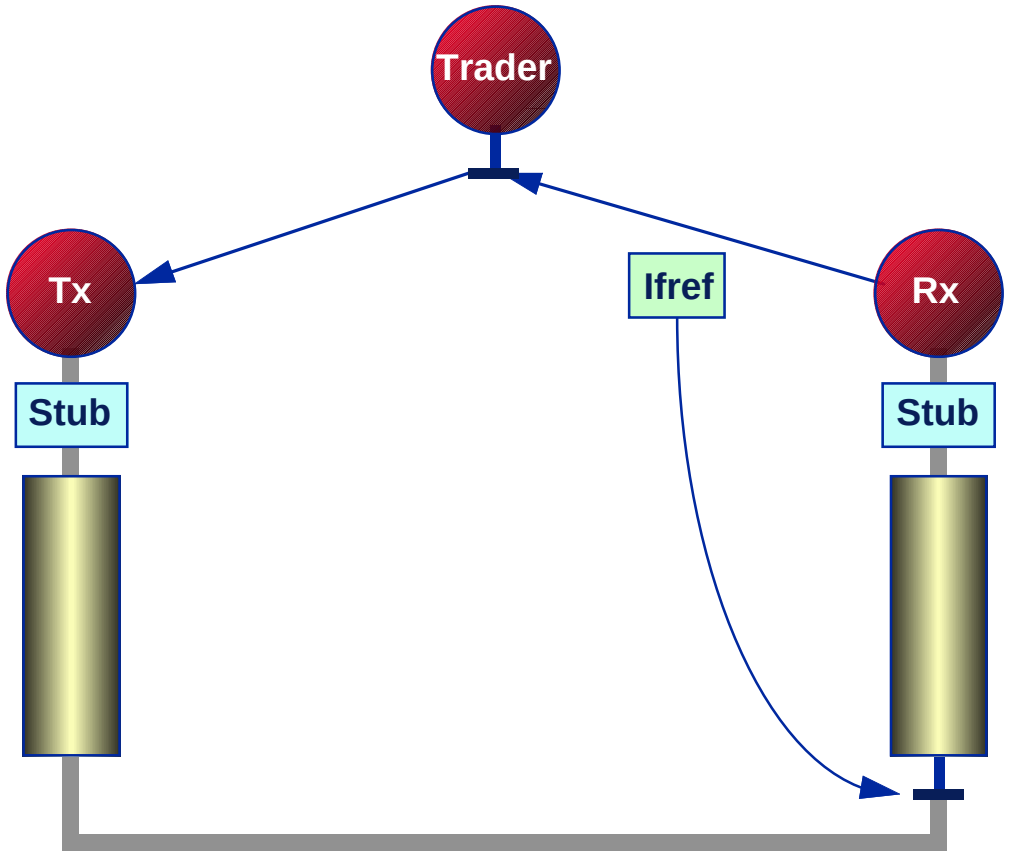
Server Endpoints



Server Endpoints



Receiver First



Transmitter First

Computational View

- Create Server or Receiver Endpoint
  - EP = Type::CreateEndpoint(Implementation)
- Create Client or Transmitter Endpoint
  - EP = Type::CreateEndpoint()
- Bind Endpoint with specific QoS
  - EP.Bind(QoS)
- Bind Endpoint with QoS at “well-known” address
  - EP.Bind(QoS, Address)

Endpoint Implementation

- Endpoints computationally visible as Invocation Refs
- Avoids introducing another computational type
- Allows EP to be treated just like any other InvRef
  - exported to Trader
  - passed out of the capsule as an operation parameter
- Any Invocation Ref may be explicitly bound

Observations

- Engineering transparency trade off at all levels
  - QoS enabled binders become protocol specific
  - applications may need to be aware of engineering mechanisms
  - ... unless hidden by QoS mapping
  - implications for dynamic loading of protocols
- Complete QoS control requires:
  - OS and network protocol support

DIMMA 2.0

- Unique, flexible ORB supporting RT and MM
- Implementation framework for
  - threading
  - resource control mechanisms
  - protocol composition
- Layered architecture
  - clean computational and engineering interfaces
  - ... allow multiple programming “personalities”
- API providing transparency between RPC and flows

DIMMA 2.0

- Explicit binding with specified QoS
  - abstract Engineering QoS binder
  - protocol specific QoS binders
- Wide range of possible QoS
  - protocol read/multiplex task policy
  - session dispatch task policy
  - channel multiplex policy
  - buffer pools and specific buffer sizes
- Dynamic protocol loading

DIMMA 2.0

Available now

- From our FTP server (ftp.ansa.co.uk)
- Directory phase3-prototypes
- Two files:
  - dimma-2.0.tar.gz (DIMMA sources)
  - dimma-tools.tar.gz (build tools)
- Runs on Solaris 2.5.1 using SPARCompiler C++

Documentation

- Available now:
  - Overview - APM.1995
  - Tracing - APM.1980
- In final update (due 15/6)
  - Build and Installation - APM.1983
  - Writing an application - APM.1987
- Design and Implementation - APM.1994 (in progress)

Other Material

- DIMMA Workshop slides:
  - Overview (APM.1894)
  - Building DIMMA (APM.1900)
  - Directories and Architecture (APM.1901)
  - Communications framework (APM.1895)
  - IIOP protocol (APM.1899)
  - Flow protocol (APM.1898)
  - Binding (APM.1897)
  - Multi-threading and locking (APM.1892)
  - ODP Library and API (APM.1887)