Patterns and Performance of Real-time Object Request Brokers

Douglas C. Schmidt

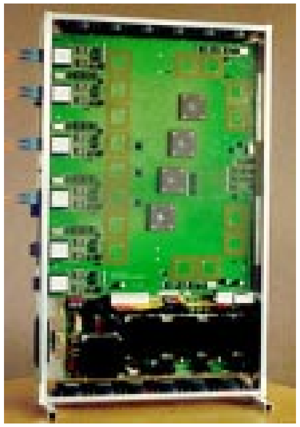
Associate Professor
schmidt@uci.edu
www.ece.uci.edu/∼schmidt/

Elec. & Comp. Eng. Dept.
University of California, Irvine
(949) 824-1901

Sponsors

NSF, DARPA, ATD, BBN, Boeing, Cisco, Comverse, GDIS, Experian, Global MT, Hughes, Kodak, Krones, Lockheed, Lucent, Microsoft, Mitre, Motorola, NASA, Nokia, Nortel, OCI, Oresis, OTI, Raytheon, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, Telcordia, USENIX

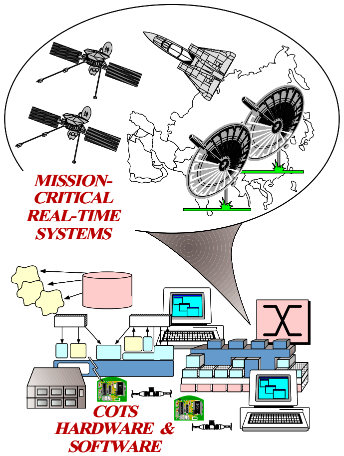
Motivation: the QoS-enabled Software Crisis



www.arl.wustl.edu/arl/

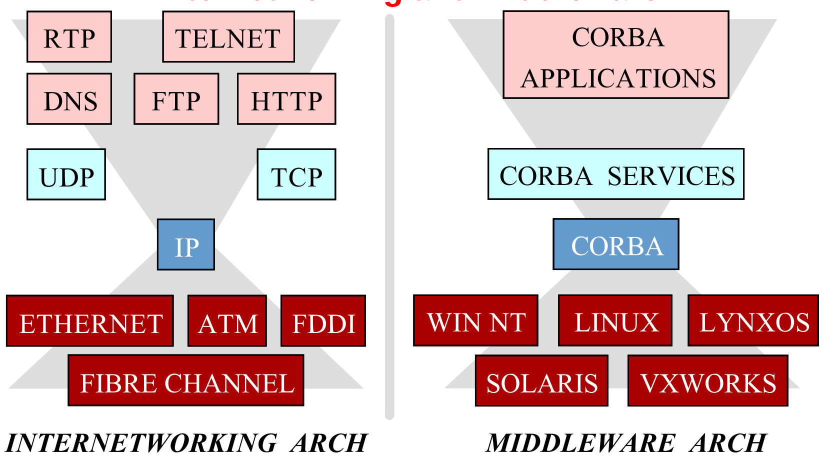
- **Symptoms**
  - Communication ***hardware*** gets smaller, faster, cheaper
  - Communication ***software*** gets larger, slower, more expensive
- **Culprits**
  - ***Inherent*** and ***accidental*** complexity
- **Solution Approach**
  - ***Standards-based COTS Hardware & Software***

Problem: the COTS Hardware & Software Crisis

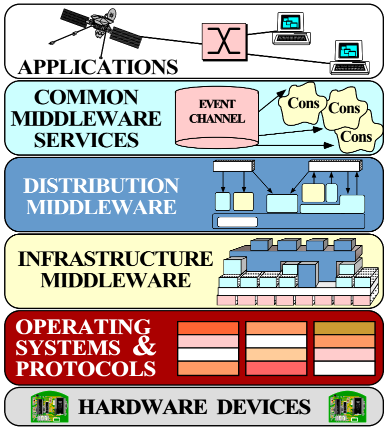


- **Context**
  - Adopting ***COTS hardware & software*** is increasingly essential for real-time mission-critical systems
- **Problems**
  - ***Inherent*** and ***accidental*** complexity
  - ***Integration*** woes
- **Solution Approach**
  - ***Standards-based adaptive COTS middleware***

Context: Levels of Abstraction in Internetworking and Middleware

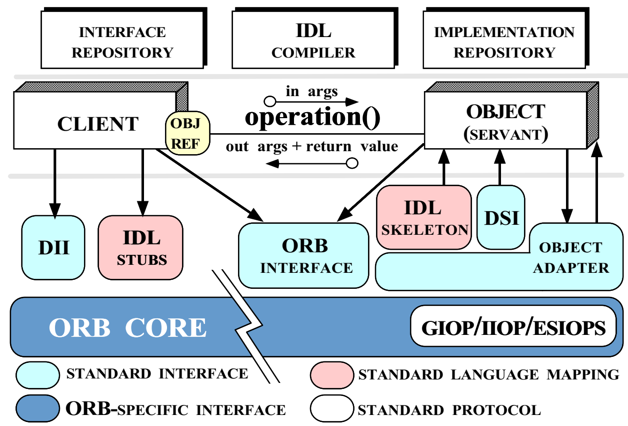


Problem: Lack of QoS-enabled Middleware



- Many applications require QoS guarantees
  - *e.g.*, avionics, telecom, WWW, medical, high-energy physics
- Building these applications manually is hard and inefficient
- Existing middleware doesn't support QoS effectively
  - *e.g.*, CORBA, DCOM, DCE, Java
- Solutions must be integrated horizontally & vertically

Candidate Solution: CORBA

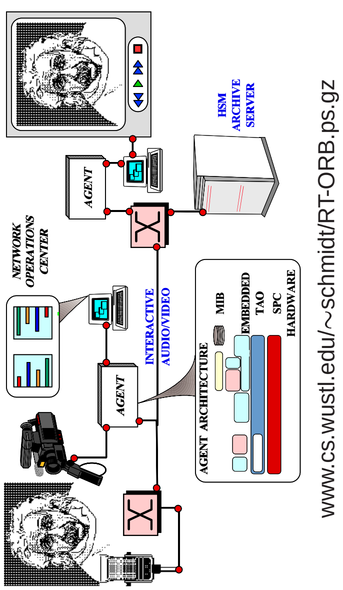


www.cs.wustl.edu/∼schmidt/corba.html

Goals of CORBA

- Simplify distribution by automating
  - Object location & activation
  - Parameter marshaling
  - Demultiplexing
  - Error handling
- Provide foundation for higher-level services

Caveat: Requirements/Limitations of CORBA for QoS-enabled Systems



www.cs.wustl.edu/∼schmidt/RT-ORB.ps.gz

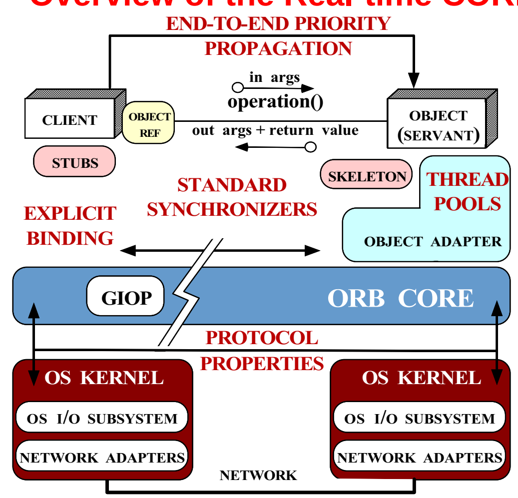
Requirements

- Location transparency
- Performance transparency
- Predictability transparency
- Reliability tranparency

Limitations

- Lack of QoS specifications
- Lack of QoS enforcement
- Lack of real-time programming features
- Lack of performance optimizations

Overview of the Real-time CORBA Specification

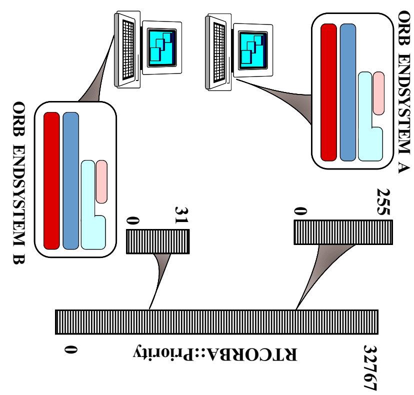


Features

1. Portable priorities
2. End-to-end priority propagation
3. Protocol properties
4. Thread pools and buffering
5. Explicit binding
6. Standard synchronizers

www.cs.wustl.edu/∼schmidt/oorc.ps.gz

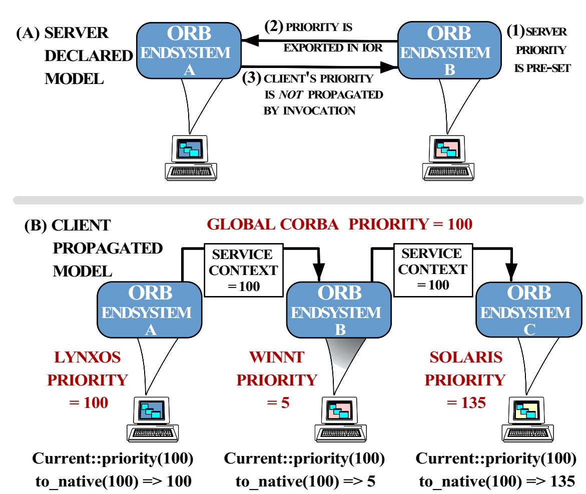
Portable Priorities



Features

- Designed to support heterogeneous real-time platforms
- *CORBA priorities* range from $0 \rightarrow 32767$
- Users can map CORBA priorities to *native OS priorities*
- No silver bullet, but rather an "enabling technique"

End-to-End Priority Propagation



Features

- Client priorities can propagate end-to-end
- Servers can also declare priority

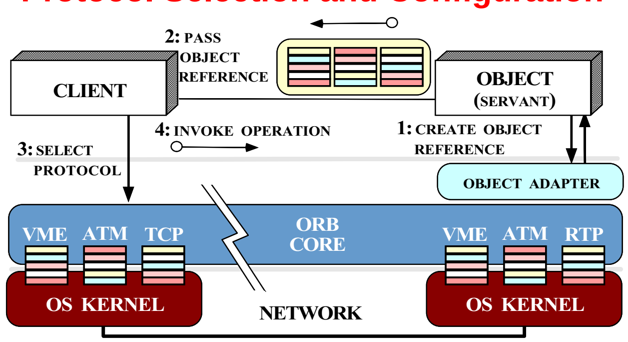
Configurable Protocol Properties

```
interface ProtocolProperties {};
typedef struct {
  IOP::ProfileId protocol_type;
  ProtocolProperties
    orb_protocol_properties;
  ProtocolProperties
    transport_protocol_properties;
} Protocol;
typedef sequence <Protocol> ProtocolList;
interface TCPProtocolProperties
  : ProtocolProperties
{
  attribute long send_buffer_size;
  attribute long recv_buffer_size;
  attribute boolean keep_alive;
  attribute boolean dont_route;
  attribute boolean no_delay;
};
```

Features

- Select and configure communication protocols
  - *e.g.,* TCP socket options
- Supports *ORB protocol* and *transport protocol* configuration
- Ordering in `ProtocolList` indicates preferences

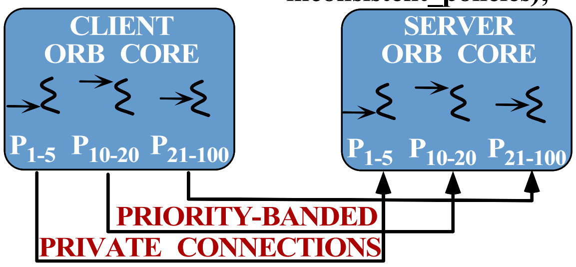
Protocol Selection and Configuration



- Protocol policies control protocol selection and configuration
  - Order of protocols indicates protocol preference
- Both server-side and client-side policies supported
  - Some policies control protocol selection, others control protocol configuration
  - Some policies are exported to client in object reference

Explicit Binding

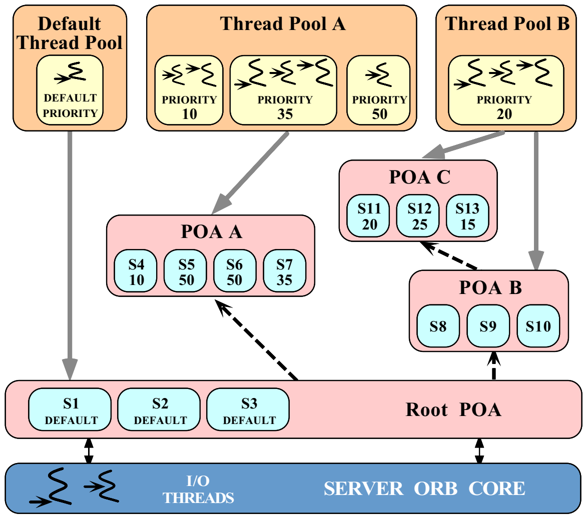
_validate_connection (out CORBA::PolicyList
inconsistent_policies);



Features

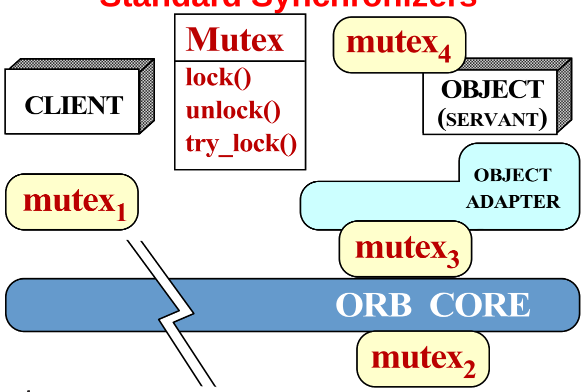
- Enables pre-establishment of connections
  - Priority-banded connections
  - Private connections
  - Protocol policies

Thread Pools



- Pre-allocate threads and thread attributes
  - Stacksize
  - Static threads and maximum threads
  - Default priority

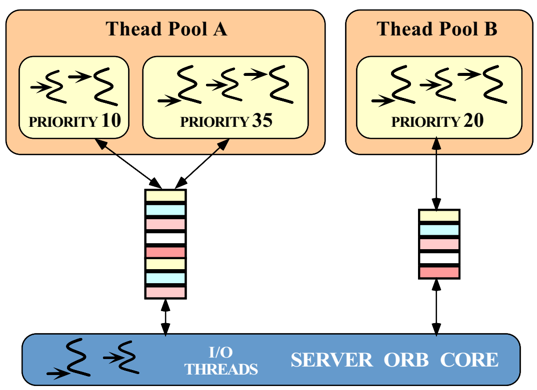
Standard Synchronizers



Features

- A portable Mutex API
  - *e.g.*, `lock`, `unlock`, `try_lock`
- Necessary to ensure consistency between ORB and application synchronizers
  - *e.g.*, priority inheritance and priority ceiling protocols
- Locality constrained

Buffering Requests



Requests are buffered when all threads are busy
Buffering can be specified in terms of:

- Number of bytes
- Number of requests

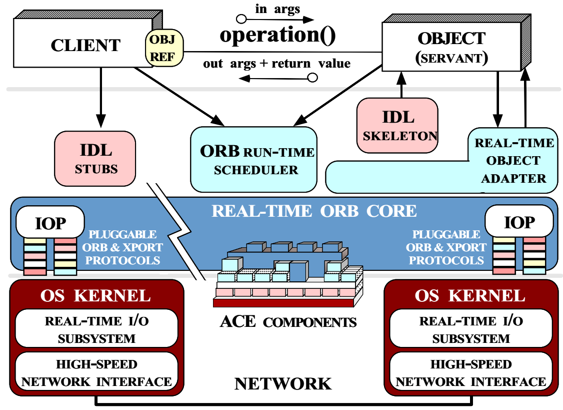
When buffers are full:

- A transient exception is thrown to client
- Request is dropped by server
- Request can be reissued later by client

Additional Information on Real-time CORBA

- Real-time CORBA 1.0 specification
  - www.cs.wustl.edu/∼schmidt/RT-ORB-std-new.pdf.gz
- Many papers at my Web site
  - www.cs.wustl.edu/∼schmidt/corba-research-realtime.html
- Upcoming OMG Real-time and Embedded CORBA Workshop
  - www.omg.org/meetings/realtime/
- Real-time ORBs
  - HighComm → www.highcomm.com
  - ORB Express → www.ois.com
  - TAO → www.theaceorb.com

Our Approach: The ACE ORB (TAO)

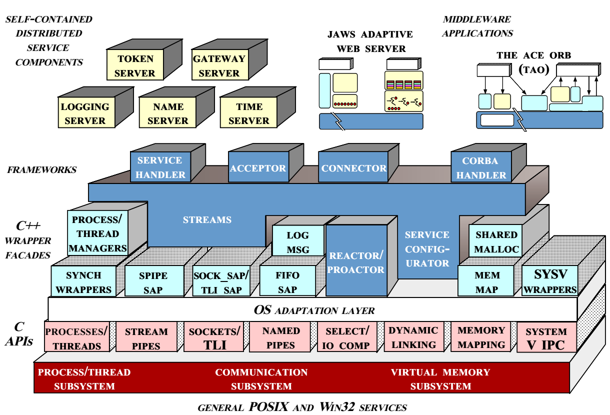


www.cs.wustl.edu/∼schmidt/TAO.html

TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX/UNIX, Win32, & RTOS platforms
  - *e.g.*, VxWorks, Chorus, LynxOS
- Leverages ACE

The ADAPTIVE Communication Environment (ACE)



www.cs.wustl.edu/∼schmidt/ACE.html

ACE Overview →

- A concurrent OO networking framework
- Available in C++ and Java
- Ported to POSIX, Win32, and RTOSs

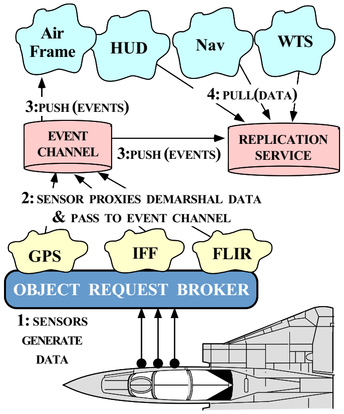
Related work →

- x-Kernel
- SysV STREAMS

ACE and TAO Statistics

- Over 50 person-years of effort
  - ACE > 200,000 LOC
  - TAO > 200,000 LOC
  - TAO IDL compiler > 130,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - ∼schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, BBN, Boeing, Ericsson, Hughes, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.theaceorb.com

Applying TAO to Avionics Mission Computing



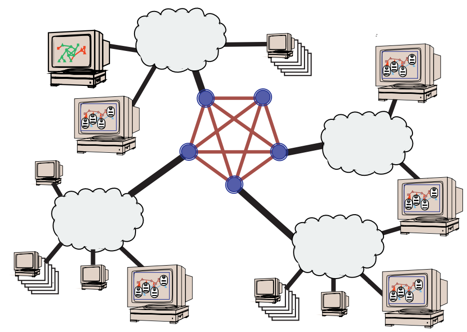
Domain Challenges

- Deterministic & statistical real-time deadlines
- Periodic & aperiodic processing
- COTS and open systems
- Reusable components
- Support platform upgrades

www.cs.wustl.edu/∼schmidt/TAO-boeing.html

www.cs.wustl.edu/∼schmidt/JSAC-98.ps.gz

Applying TAO to Distributed Interactive Simulations



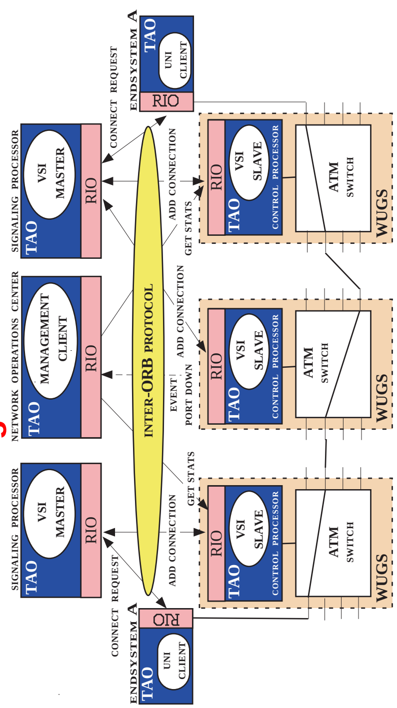
www.cs.wustl.edu/∼schmidt/Words99.ps.gz

Domain Challenges

- High scalability and group communication
- High throughput and low latency
- “Interactive” real-time
- Multi-platform

hlasdc.dmso.mil/RTISUP/hla_soft/hla_soft.htm

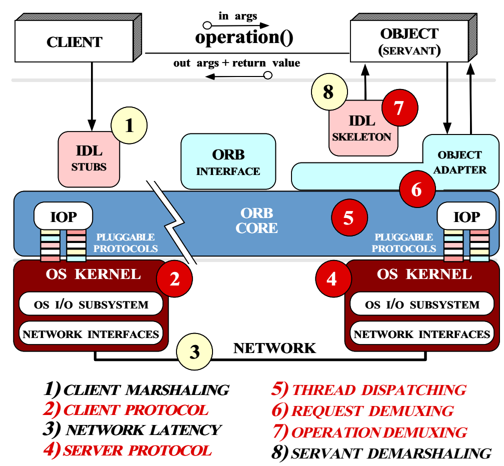
Applying TAO to Embedded Network Element Management and Control



Domain Challenges

- High-speed (20 Gbps) ATM switches w/embedded controllers
- Low-latency and statistical real-time deadlines
- COTS infrastructure, standards-based open systems, and small footprint

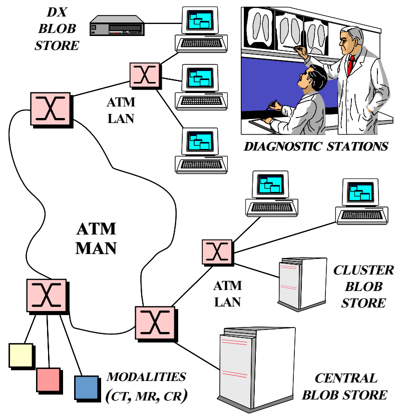
Optimization Challenges for QoS-enabled ORBs



Key Challenges

- Alleviate priority inversion and non-determinism
- Reduce demultiplexing latency/jitter
- Ensure protocol flexibility
- Specify QoS requirements
- Schedule operations
- Eliminate (de)marshaling overhead
- Minimize footprint

Problem: Optimizing Complex Software



www.cs.wustl.edu/~schmidt/JSAC-99.ps.gz

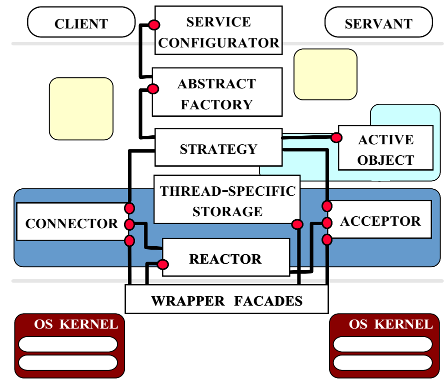
Common Problems →

- Optimizing complex software is hard
- Small "mistakes" can be costly

Solution Approach (Iterative) →

- Pinpoint overhead via *white-box* metrics
  - *e.g.*, `Quantify` and `VMEtro`
- Apply patterns and framework components
- Revalidate via white-box and black-box metrics

Solution 1: Patterns and Framework Components



www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz

Definitions

- *Pattern*
  - A solution to a problem in a context
- *Framework*
  - A "semi-complete" application built with components
- *Components*
  - Self-contained, "pluggable" ADTs

Solution 2: ORB Optimization Principle Patterns

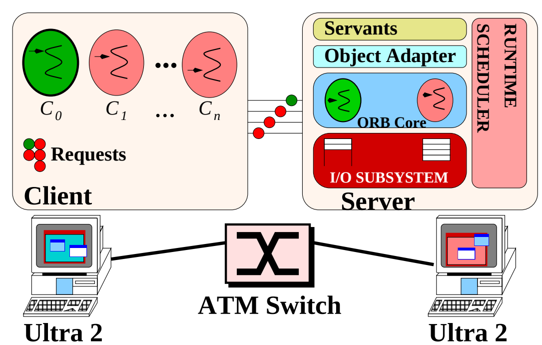
Definition

- ***Optimization principle patterns*** document rules for avoiding common design and implementation problems that can degrade the efficiency, scalability, and predictability of complex systems

Optimization Principle Patterns Used in TAO

#	Optimization Principle Pattern
1	Optimize for the common case
2	Remove gratuitous waste
3	Replace inefficient general-purpose functions with efficient special-purpose ones
4	Shift computation in time, *e.g.*, precompute
5	Store redundant state to speed-up expensive operations
6	Pass hints between layers and components
7	Don't be tied to reference implementations/models
8	Use efficient/predictable data structures

ORB Latency and Priority Inversion Experiments

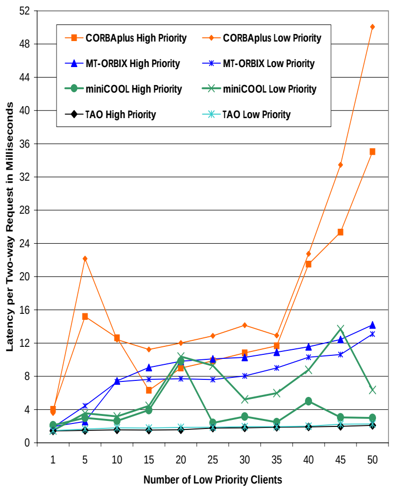


www.cs.wustl.edu/~schmidt/RT-perf.ps.gz

Method

- Vary ORBs, hold OS constant
- Solaris real-time threads
- High priority client C_0 connects to servant S_0 with matching priorities
- Clients $C_1 \ldots C_n$ have same lower priority
- Clients $C_1 \ldots C_n$ connect to servant S_1
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

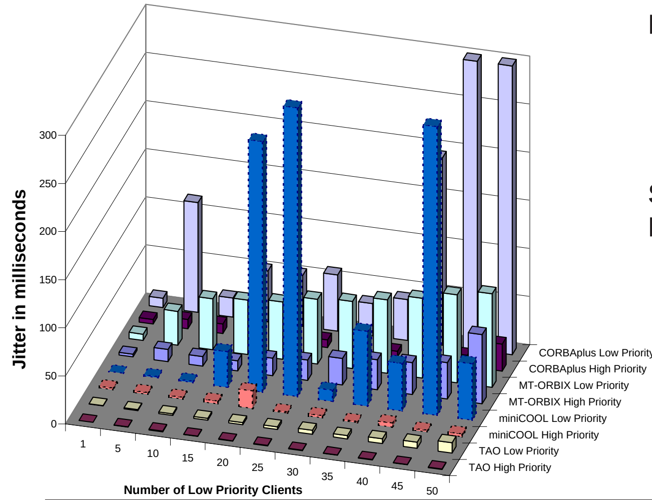
ORB Latency and Priority Inversion Results



Synopsis of Results

- TAO's latency is lowest for large # of clients
- TAO avoids priority inversion
  - *i.e.*, high priority client always has lowest latency
- Primary overhead stems from *concurrency* and *connection* architecture
  - *e.g.*, synchronization and context switching

ORB Jitter Results



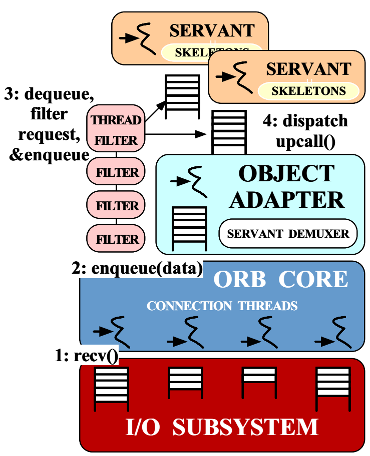
Definition

- Jitter → standard deviation from average latency

Synopsis of Results

- TAO's jitter is lowest and most consistent
- CORBAplus' jitter is highest and most variable

Problem: Improper ORB Concurrency Models

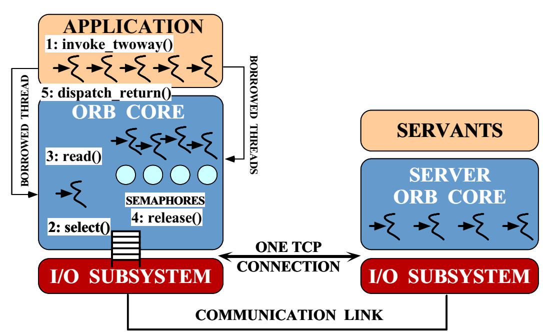


Common Problems

- High context switching and synchronization overhead
- Thread-level and packet-level priority inversions
- Lack of application control over concurrency model

www.cs.wustl.edu/∼schmidt/CACM-arch.ps.gz

Problem: ORB Shared Connection Models

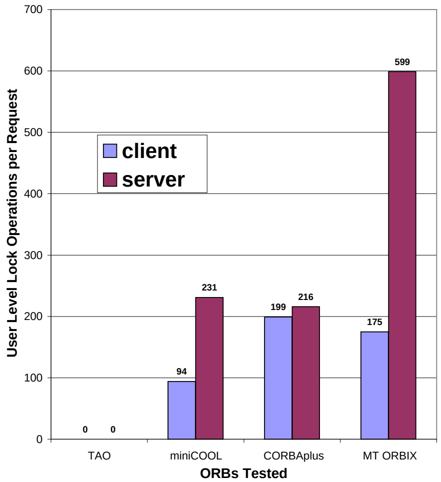


Common Problems

- Request-level priority inversions
  - Sharing multiple priorities on a single connection
- Complex connection multiplexing
- Synchronization overhead

www.cs.wustl.edu/∼schmidt/RTAS-98.ps.gz

Problem: High Locking Overhead

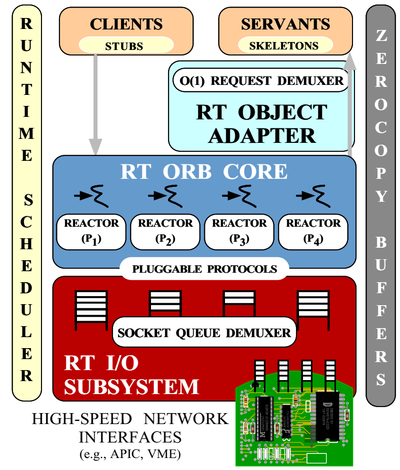


Common Problems

- Locking overhead affects latency and jitter significantly
- Memory management commonly involves locking

www.cs.wustl.edu/∼schmidt/RTAS-98.ps.gz

Solution: TAO's ORB Endsystem Architecture



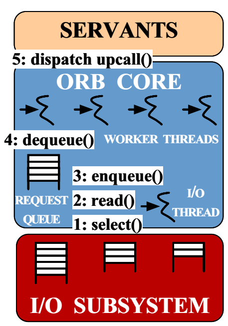
Solution Approach →

- Integrate scheduler into ORB endsystem
- Co-schedule threads
- Leader/followers thread pool

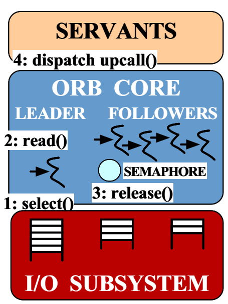
Principle Patterns →

- Pass hints, precompute, optimize common case, remove gratuitous waste, store state, don't be tied to reference implementations & models

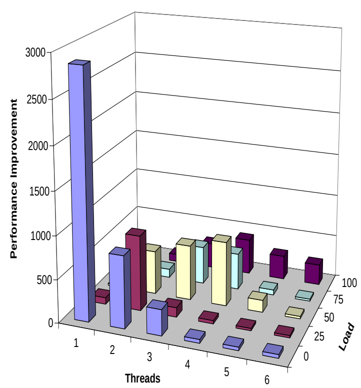
Thread Pool Comparison Results



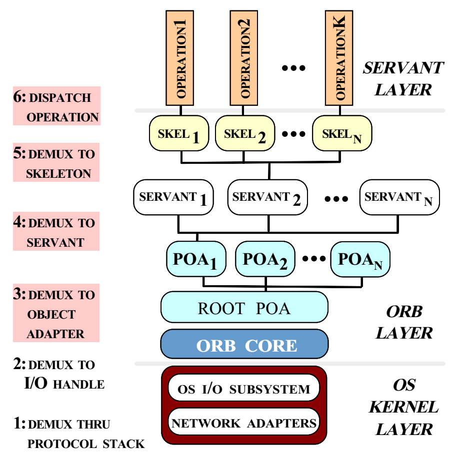
Worker Thread Pool



Leader/Follower Thread Pool



Problem: Reducing Demultiplexing Latency

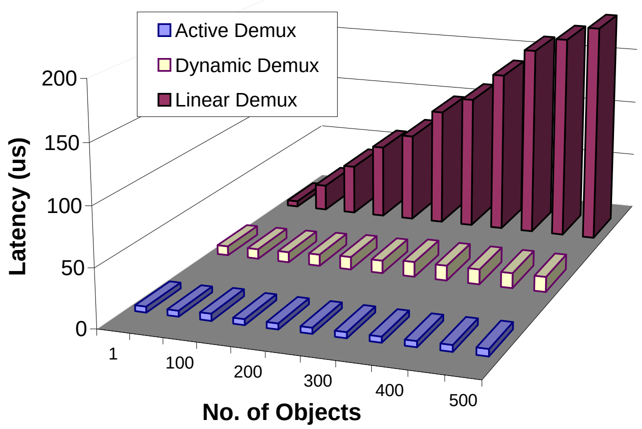


Design Challenges

- Minimize demuxing layers
- Provide $O(1)$ operation demuxing through all layers
- Avoid priority inversions
- Remain CORBA-compliant

www.cs.wustl.edu/∼schmidt/POA.ps.gz

Servant Demultiplexing Results



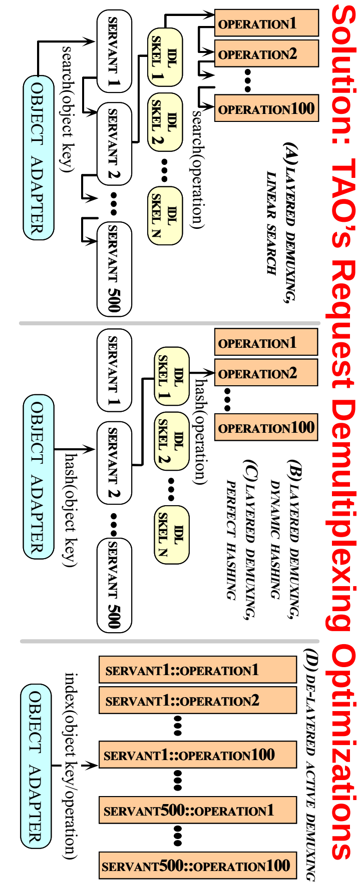
Synopsis of Results

- Linear demux is costly
- Active demux is most efficient & predictable

Principle Patterns

- Precompute, pass hints, use special-purpose & predictable data structures

Solution: TAO's Request Demultiplexing Optimizations



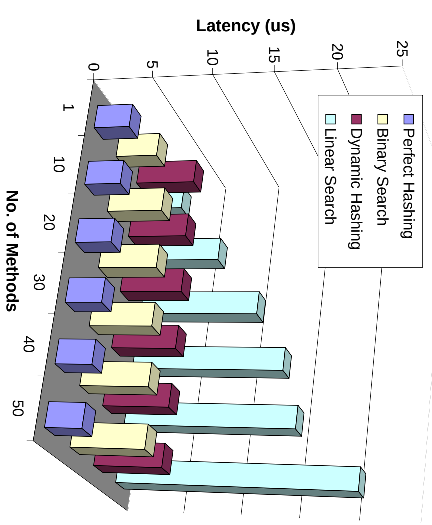
Demuxing

- www.cs.wustl.edu/∼schmidt/{ieee_tc-97,COOTS-99}.ps.gz

Perfect hashing

- www.cs.wustl.edu/∼schmidt/gperf.ps.gz

Operation Demultiplexing Results



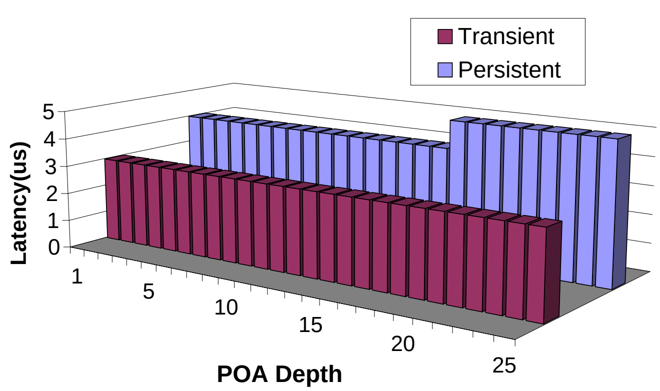
Synopsis of Results →

- Perfect Hashing
  - Highly predictable
  - Low-latency
- Others strategies slower

Principle Patterns →

- Precompute, use predictable data structures, remove gratuitous waste

POA Demultiplexing Results



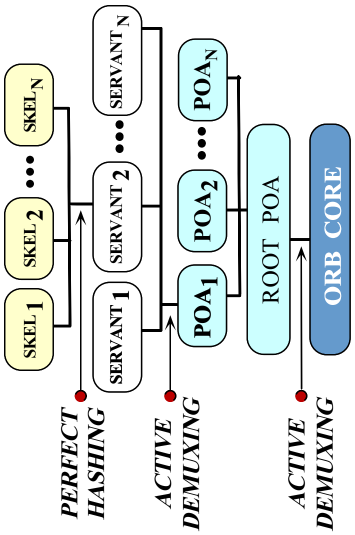
Synopsis of Results

- Active demux is efficient & predictable for both transient and persistent object references.

Principle Patterns

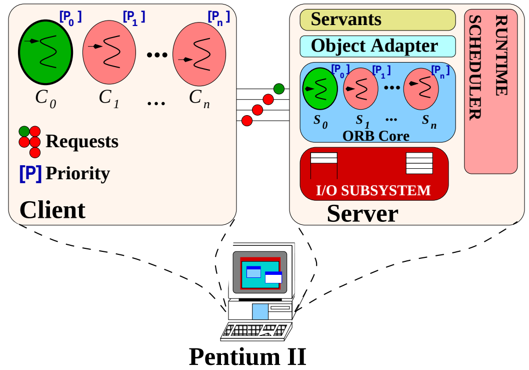
- Precompute, pass hints, use special-purpose & predictable data structures, ignore ref models

TAO Request Demultiplexing Summary



Demultiplexing Stage	Absolute Time (μs)
1. Request parsing	2
2. POA demux	2
3. Servant demux	3
4. Operation demux	2
5. Parameter demarshaling	operation dependent
6. User upcall	servant dependent
7. Results marshaling	operation dependent

Real-time ORB/OS Performance Experiments

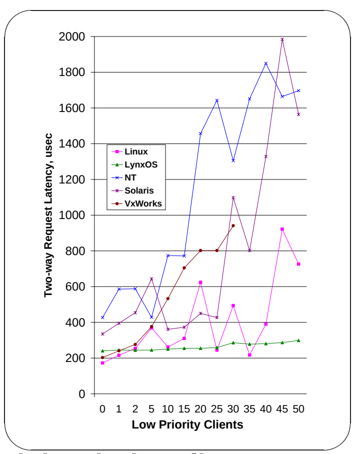


www.cs.wustl.edu/~schmidt/RT-OS.ps.gz

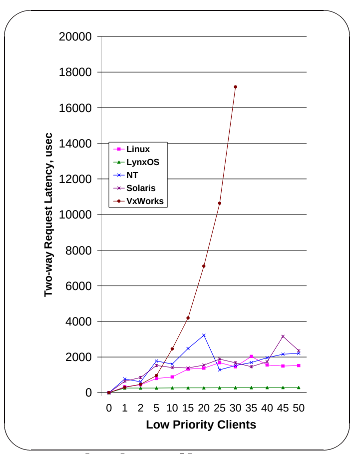
Method

- Vary OS, hold ORBs constant
- Single-processor Intel Pentium II 450 Mhz, 256 Mbytes of RAM
- Client and servant run on the same machine
- Client C_i connects to servant S_i with priority P_i
  - i ranges from $1 \ldots 50$
- Clients invoke two-way CORBA calls that cube a number on the servant and returns result

Real-time ORB/OS Performance Results

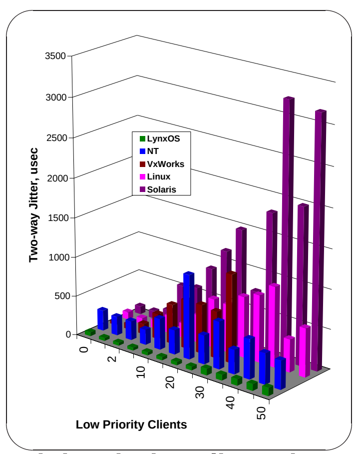


High-priority Client Latency

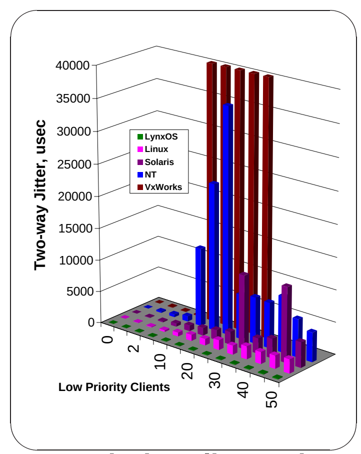


Low-priority Clients Latency

Real-time ORB/OS Jitter Results

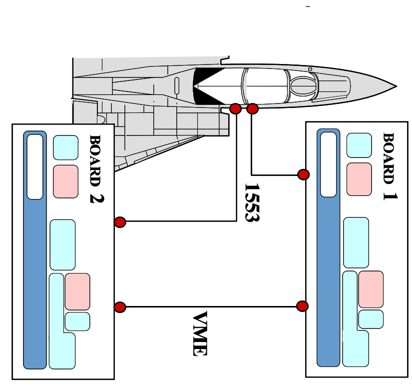


High-priority Client Jitter



Low-priority Clients Jitter

Problem: Hard-coded ORB Messaging and Transport Protocols



- GIOP/IIOP are not sufficient, *e.g.*:
  - GIOP message footprint may be too large
  - TCP lacks necessary QoS
  - Legacy commitments to existing protocols
- Many ORBs do not support "pluggable protocols"
  - This makes ORBs inflexible and inefficient

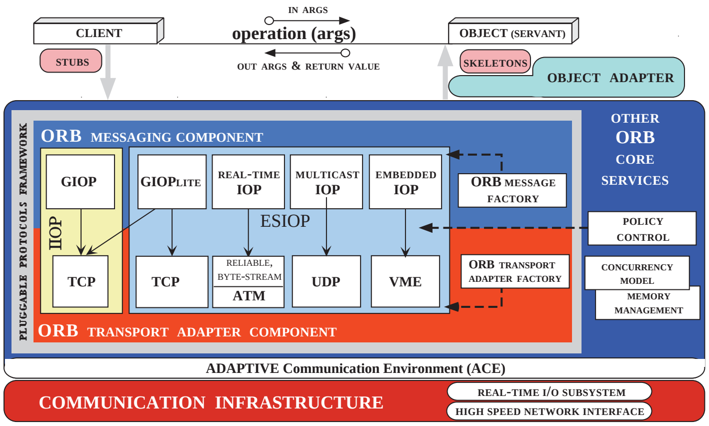
One Solution: Hacking GIOP

- GIOP requests include fields that aren't needed in homogeneous embedded applications
  - *e.g.*, GIOP magic #, GIOP version, byte order, request principal, etc.
- These fields can be omitted without any changes to the standard CORBA programming model
- TAO's `-ORBgioplite` option save 15 bytes per-request, yielding these calls-per-second:

	Marshaling-enabled			Marshaling-disabled		
	min	max	avg	min	max	avg
GIOP	2,878	2,937	2,906	2,912	2,976	2,949
GIOPlite	2,883	2,978	2,943	2,911	3,003	2,967

- The result is a measurable improvement in throughput/latency
  - However, it's so small (2%) that hacking GIOP is of minimal gain except for low-bandwidth links

Better Solution: TAO's Pluggable Protocols Framework



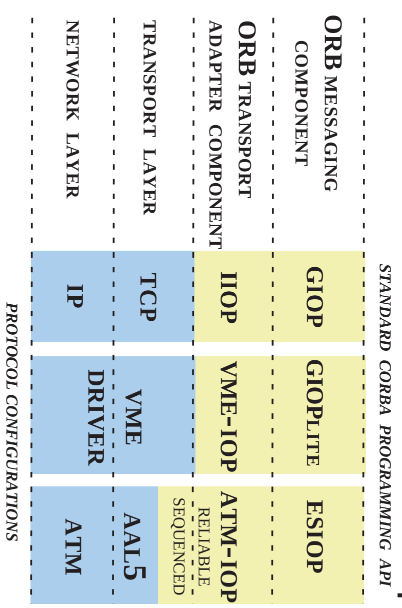
Features

- Pluggable *ORB messaging* and *transport* protocols
- Highly efficient and predictable behavior

Principle Patterns

- Replace general-purpose functions (protocols) with special-purpose ones

CORBA Protocol Interoperability Architecture

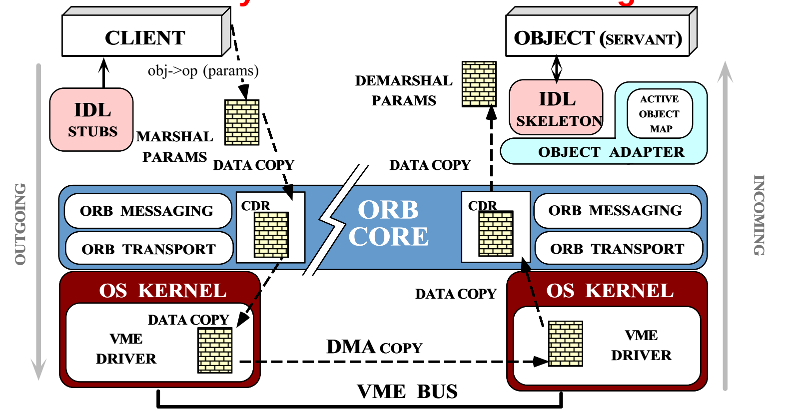


Features →

- Presentation layer
  - *e.g.*, CDR
- Message formats
  - *e.g.*, GIOP, DCE
- Transport assumptions
  - *e.g.*, TCP, ATM
- Object addressing
  - *e.g.*, IIOP IOR

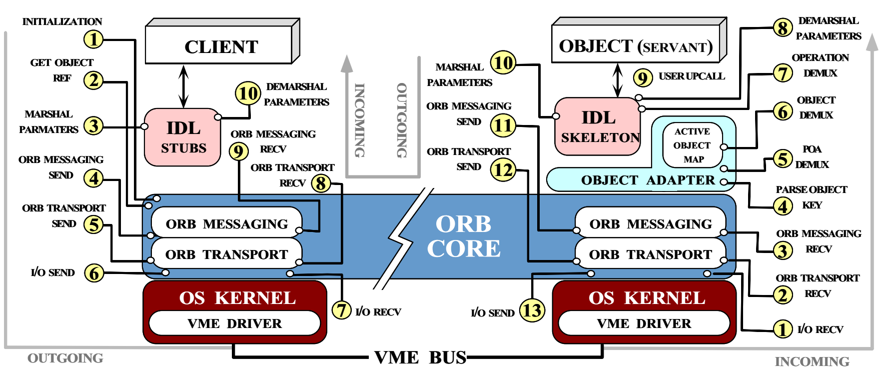
www.cs.wustl.edu/∼schmidt/pluggable_protocols.ps.gz

Embedded System Benchmark Configuration



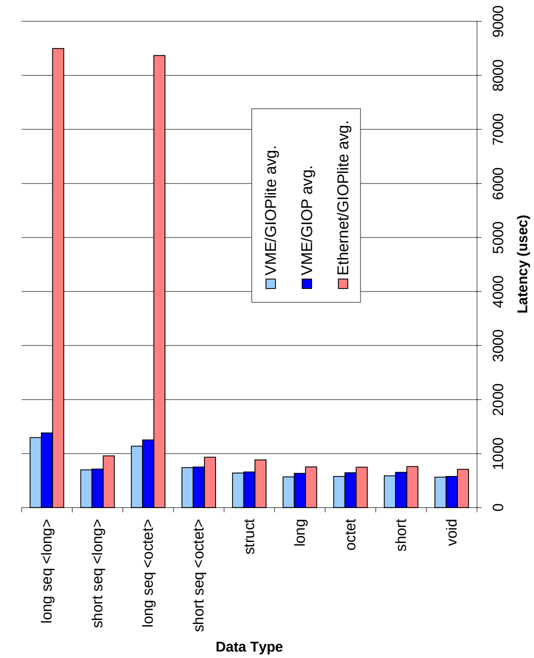
VxWorks running on 200 Mhz PowerPC over 320 Mbps VME & 10 Mbps Ethernet

Pinpointing ORB Overhead with VMEtro Timeprobes



- Timeprobes use VMEtro monitor, which measures end-to-end time
- Timeprobe overhead is minimal, *i.e.*, 1 μsec

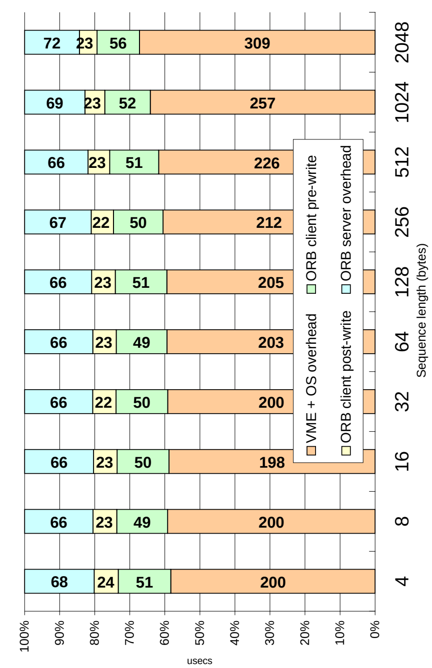
Ethernet & VME Two-way Latency Results



Synopsis of Results

- VME protocol is much faster than Ethernet
- No application changes are required to support VME

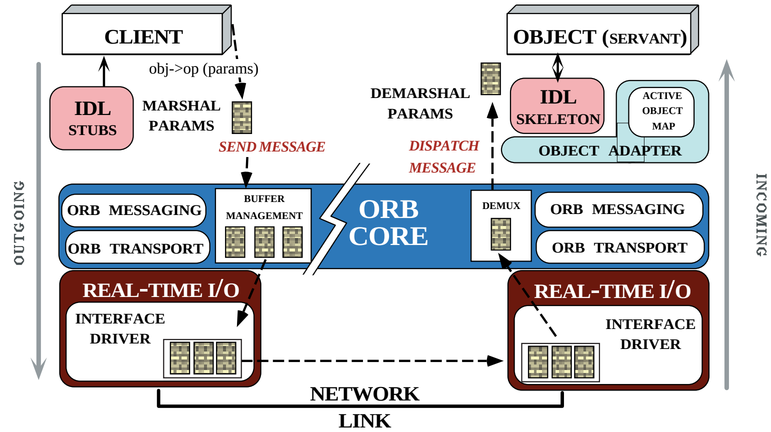
ORB & VME One-way Overhead Results



Synopsis of Results

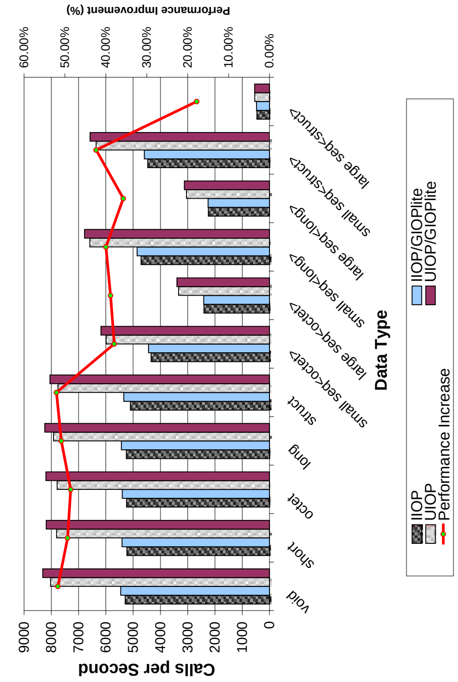
- ORB overhead is relatively constant and low
  - *e.g.*, ~110 μsecs per end-to-end operation
- Bottleneck is VME driver and OS, not ORB

Workstation Benchmark Configuration



Debian Linux running on 400 Mhz workstation over Local IPC

Blackbox Two-way Latency Results



Synopsis of Results

- Local IPC more efficient than TCP/IP over loopback
- No application changes are required to support multiple protocols

Client Whitebox Latency Results

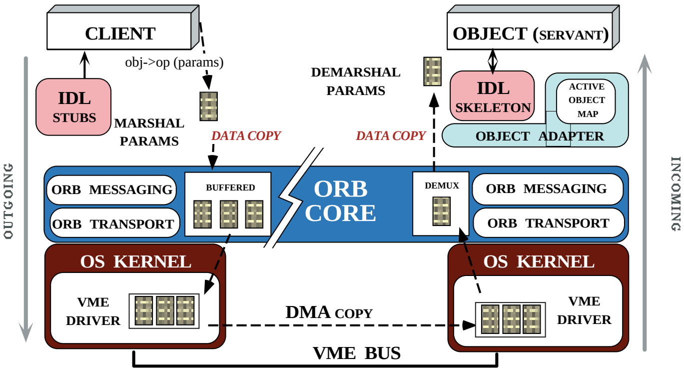
Direction	Client Activities	Absolute Time (μs)
Outgoing	1. *Initialization*	6.30
	2. *Get object reference*	15.6
	3. *Parameter marshal*	0.74 (param. dependent)
	4. *ORB messaging send*	7.78
	5. *ORB transport send*	1.02
	6. *I/O*	8.70 (op. dependent)
	7. *ORB transport recv*	50.7
	8. *ORB messaging recv*	9.25
	9. *Parameter demarshal*	op. dependent

Xeon platform is quad-CPU 400 Mhz with 1 Gigabytes RAM

Server Whitebox Latency Results on Xeon/NT

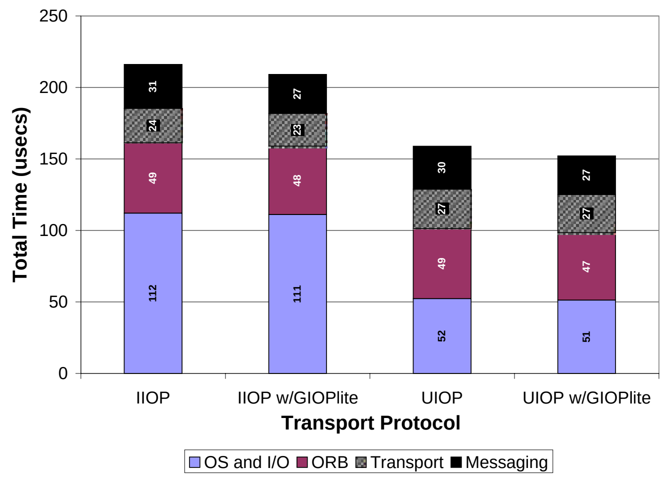
Direction	Server Activities	Absolute Time (μs)
Incoming	1. *I/O*	7.0 (op. dependent)
	2. *ORB transport recv*	24.8
	3. *ORB messaging recv*	4.5
	4. *Parsing object key*	4.6
	5. *POA demux*	1.39
	6. *Servant demux*	4.6
	7. *Operation demux*	4.52
	8. *User upcall*	3.84 (op. dependent)
Outgoing	9. *ORB messaging send*	4.56
	10. *ORB transport send*	93.6

One-Way Delayed Buffering Strategy



- Copy params to new buffer
- Requests buffered in the Transport Adaptor
- Flush at byte count or timeout
- Send as one ORB message but multiple requests
- Server demultiplexes individual requests

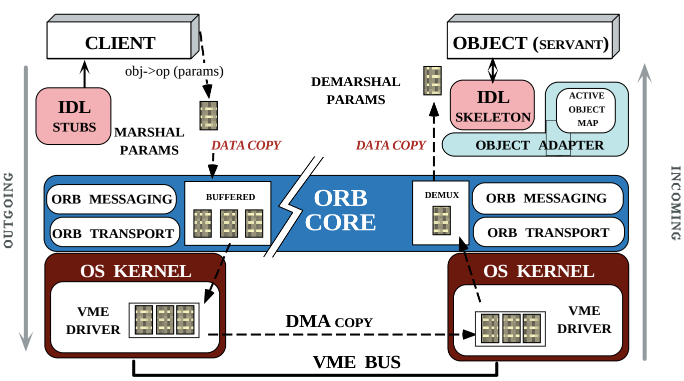
ORB & Transport Overhead Results



Synopsis of Results

- ORB overhead is relatively constant and low
  - *e.g.*, $\sim$49 μsecs per two-way operation
- Bottleneck is OS and I/O operation

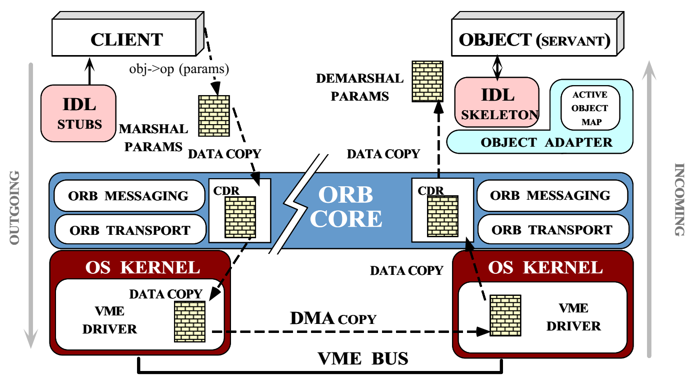
Shared Buffer Strategy



- Request free buffer
- Add to Send queue
- Return to Free pool

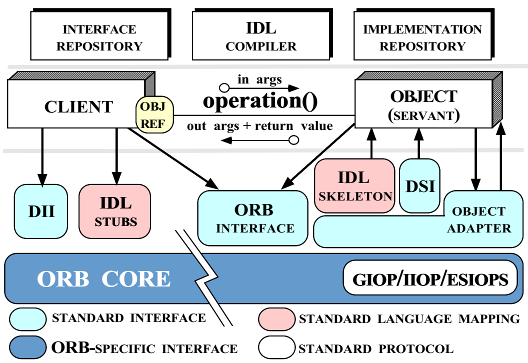
- Request free buffer
- Add to Revc queue
- Return to Free pool

Data Copies in the Pluggable Protocols



- Marshal parameters, data copy to CDR stream
- VME send, data copy from CDR stream to VME buffers
- DMA, data copy over VME Bus
- VME read, data copy to CDR stream
- Demarshal parameters, data copy to method parameters

Problem: Overly Large Memory Footprint



www.cs.wustl.edu/~schmidt/COOTS-99.ps.gz

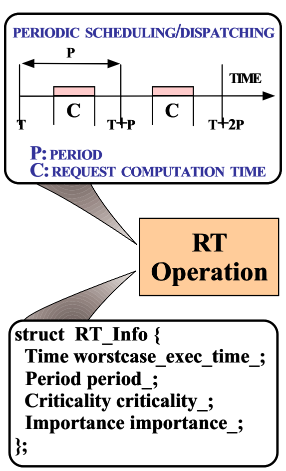
- **Problem**
  - ORB footprint is too big for some embedded apps
- **Unnecessary Features**
  - DSI, DII, & Dynamic Any
  - Interface Repository
  - Advanced POA features
  - CORBA/COM interworking

Solution: Minimum CORBA

Component	CORBA	Minimum CORBA	Percentage Reduction
POA	282k	207k	26.5
ORB Core	347k	330k	4.8
Dynamic Any	131k	0	100
CDR Interpreter	69k	69k	0
IDL Compiler	10k	11k	0
Pluggable Protocols	15k	15k	0
Default Resources	8k	8k	0
Total	862k	640k	25.8

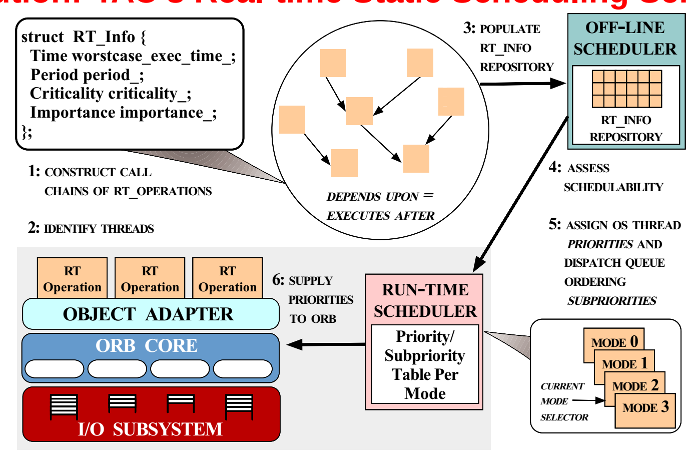
Applying Minimum CORBA subsetting to TAO reduces memory footprint by ~25% (on SPARC with EGCS) and increases ORB determinism

Problem: Providing QoS to CORBA Operations



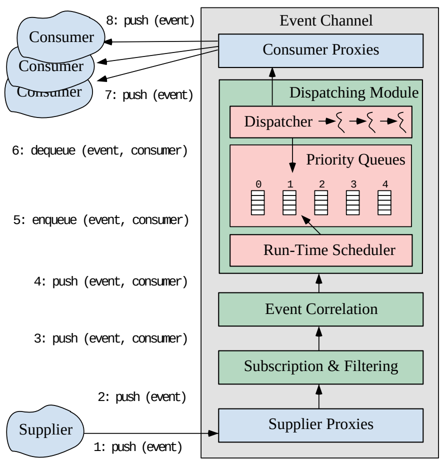
- **Design Challenges**
  - Specifying/enforcing QoS requirements
  - Focus on *Operations* upon *Objects*
    - ∗ Rather than on communication channels or threads/synchronization
  - Support static *and* dynamic scheduling
- **Solution Approach**
  - Servants publish resource (*e.g.*, CPU) requirements and (periodic) deadlines
  - Most clients are also servants

Solution: TAO's Real-time Static Scheduling Service



www.cs.wustl.edu/~schmidt/TAO.ps.gz

TAO's RT Event Service Architecture

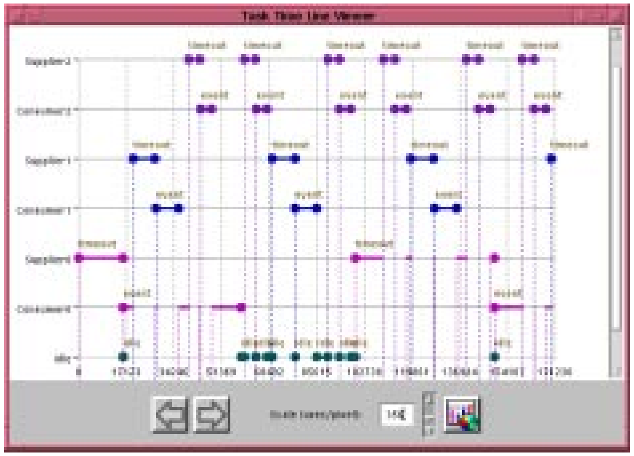


Features →

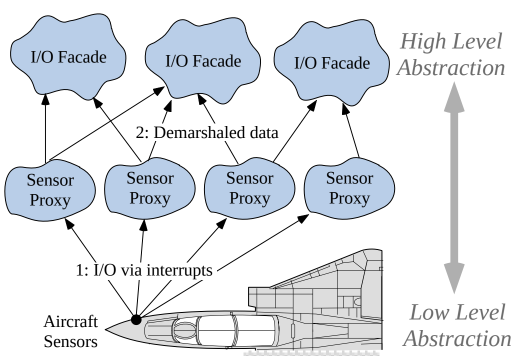
- Integrated with RT Scheduler
- Stream-based architecture
  - Enhance pluggability
- Source and type-based filtering
- Event correlations
  - *Conjunctions* (A+B+C)
  - *Disjunctions* (A|B|C)

www.cs.wustl.edu/∼schmidt/JSAC-98.ps.gz

Visualizing Periods for Avionics Operations



Example: Applying TAO to Real-time Avionics

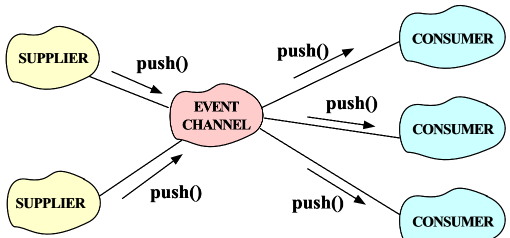


- **Synopsis**
  - *Typical Interactions*
    - ∗ I/O arrives
    - ∗ Proxies demarshal data
    - ∗ Facades process data
  - *Advantages:*
    - ∗ Efficient control flow
    - ∗ Clean layered architecture
  - *Disadvantages:*
    - ∗ Coupled layers
    - ∗ Inflexible scheduling

Forces/Domain Characteristics

- I/O driven
  - Periodic processing requirements
- Complex dependencies
  - *e.g.*, I/O Facades depend on multiple sensor proxies
- Real-time constraints
  - Deterministic and statistical deadlines
  - Static scheduling (*e.g.*, rate monotonic)
- Single-Processor (VxWorks)
  - Single address space
  - No distribution requirements (yet)

Candidate Solution: COS Event Service



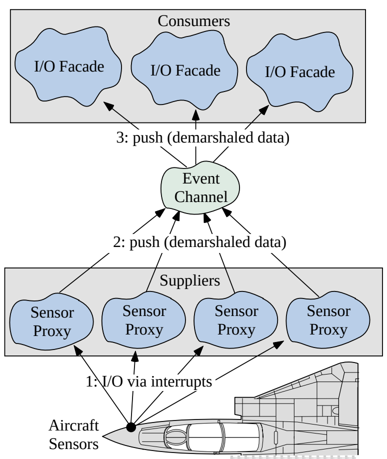
www.cs.wustl.edu/~schmidt/report-doc.html

- **Features**
  – Decoupled consumers and suppliers
  – Transparent group communication
  – Asynchronous communication
  – Abstraction for distribution
  – Abstraction for concurrency

Applying the COS Event Service to Real-time Avionics

- TAO is currently used at Boeing for avionics mission computing
  – Initial flight dates are mid-summer 1998
- Extensive benchmarks demonstrate it is possible to meet stringent performance goals with real-time CORBA
  – *e.g.*, for Boeing, target latency for CORBA oneway operations is 150 μsecs for 100 Mhz PowerPC running over MVME 177 boards
- Technology transfer to commercial vendors via OMG RT SIG and DARPA Quorom program & OCI

Overview of Avionics Mission Computing

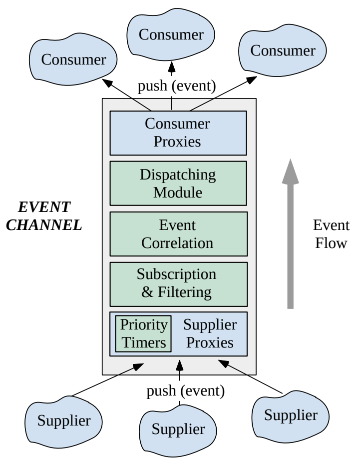


- **Typical Interactions**
  – I/O arrives
  – Proxies demarshal data
  – Proxies push to channel
  – EC pushes to facades
  – Facades process data
- **Advantages:**
  – Anonymous consumers/suppliers
  – Group communication
  – Asynchronous pushes

Issues Not Addressed by COS Event Service

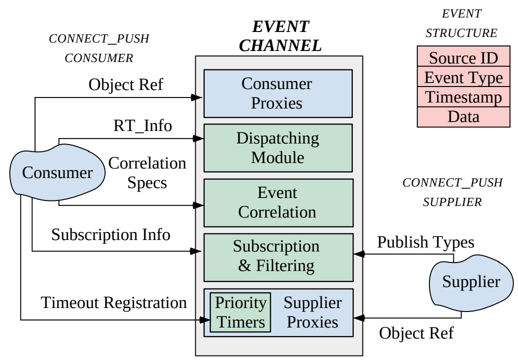
- No support for complex event dependencies
  – Consumer-specified event filtering
  – Event correlations (*e.g.*, waiting for events A *and* B before pushing)
- No support for real-time scheduling policies
  – Priority-based dispatching (*e.g.*, which consumer is dispatched first)
  – Priority-based preemption policies and mechanisms
  – Interval timeouts for periodic processing
  – Deadline timeouts for “failed” event dependencies

TAO's Event Service Architecture



- **Features**
  - Stream-based architecture
    - Enhance pluggability
  - Subscription/filtering
    - Source and type-based filtering
  - Event correlations
    - Conjunctions (A+B+C)
    - Disjunctions (A|B|C)

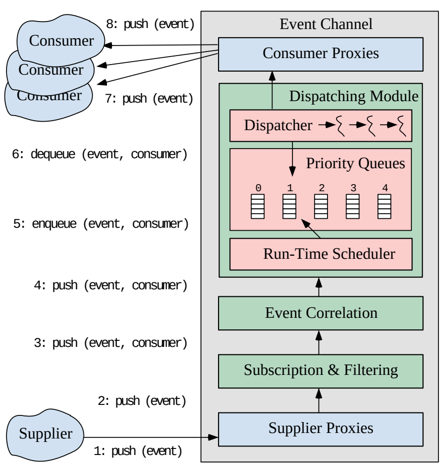
Collaborations in the RT Event Channel



- Well-defined event structure
  - CORBA Anys are inefficient
- Augmented COS interfaces:
  - Extra QoS structure to connect suppliers and consumers

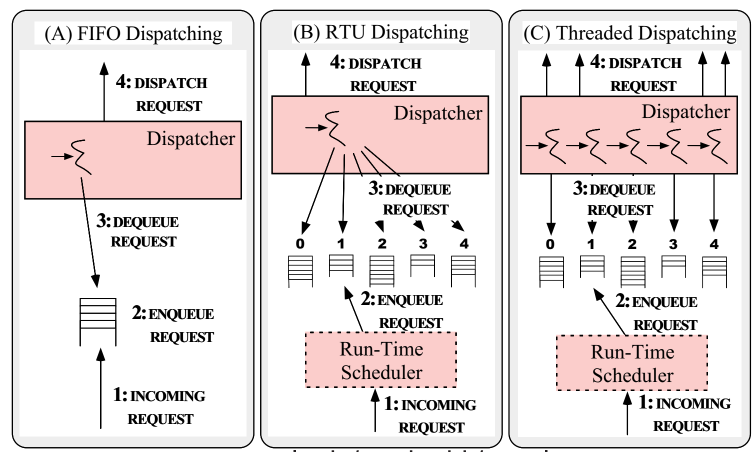
www.cs.wustl.edu/~schmidt/events_tutorial.html

Real-Time Event Dispatching with TAO's Event Service



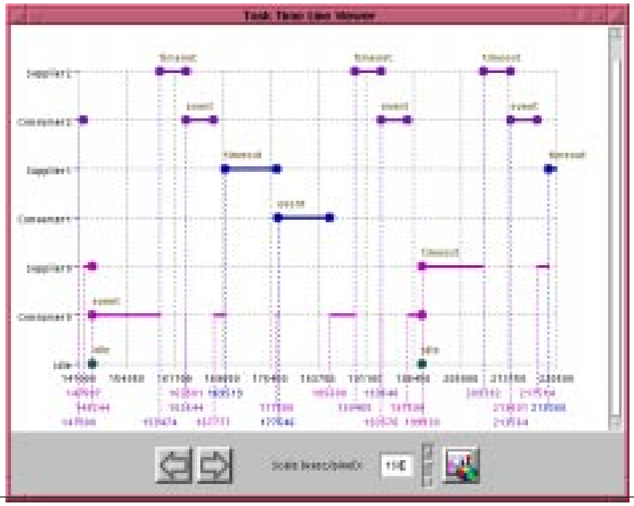
- **Features**
  - Run-time scheduler
    - Determines event priority
  - 2-level priority queues
    - Preemption groups
    - Priority queues
  - Dispatcher
    - Encapsulates concurrency policy

Real-time Event Channel Dispatching Experiments

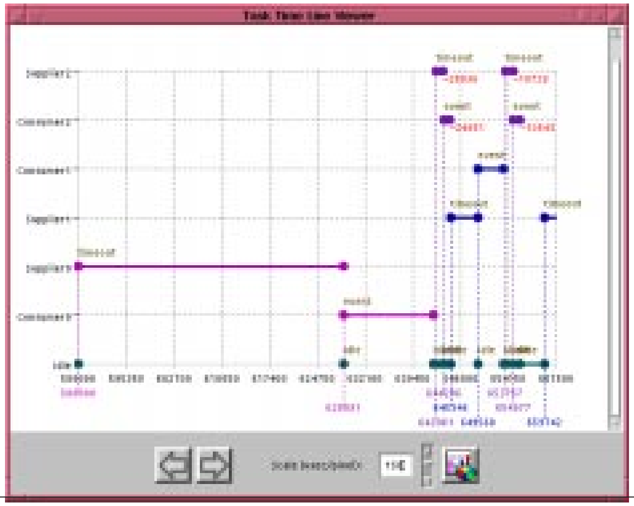


www.cs.wustl.edu/~schmidt/oopsla.ps.gz

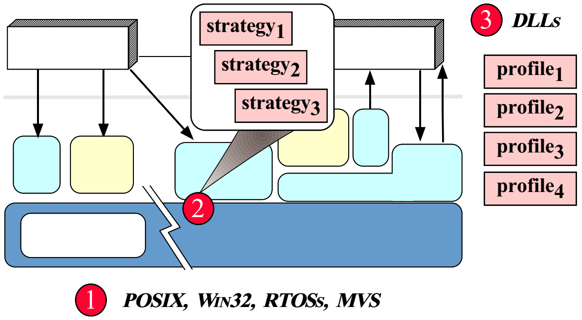
Multi-Threaded Dispatching



Single-Threaded Dispatching

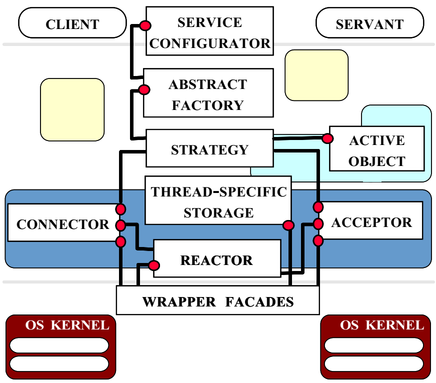


Dimensions of ORB Extensibility



1. Extensible to retargeting on new platforms
2. Extensible via custom implementation strategies
3. Extensible via dynamic configuration of custom strategies

Applying Patterns to Develop Extensible ORBs



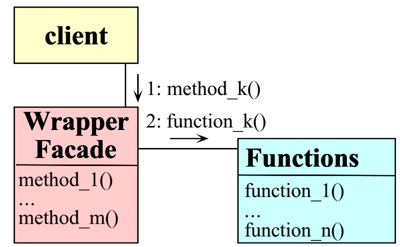
www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz

- *Factories* produce *Strategies*
- *Strategies* implement interchangeable policies
- Concurrency strategies use *Reactor* and *Active Object*
- *Acceptor-Connector* decouple transport from GIOP operations
- *Service Configurator* permits dynamic configuration

Addressing ORB Portability and Typesafety Challenges

- Problem
  - Building an ORB using low-level system APIs is hard
- Forces
  - Low-level APIs are tedious to program
  - Low-level APIs are error-prone
  - Low-level APIs are non-portable
- Solution
  - Apply the *Wrapper Facade* pattern to encapsulate low-level OS programming details

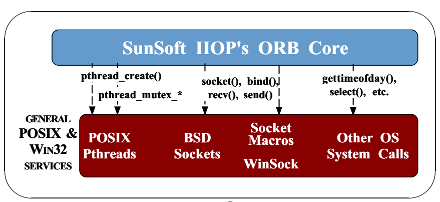
Enhancing Portability and Typesafety with the Wrapper Facade Pattern



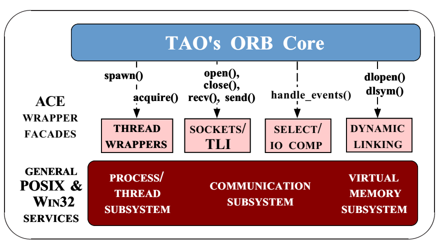
www.cs.wustl.edu/∼schmidt/
wrapper-facade.ps.gz

- **Intent**
  - *Encapsulates low-level system calls within type-safe, modular, and portable class interfaces*
- **Forces Resolved**
  - Avoid tedious, error-prone, and non-portable system APIs
  - Create cohesive abstractions
  - Don't compromise performance

Using the Wrapper Facade Pattern in TAO



SunSoft IIOP



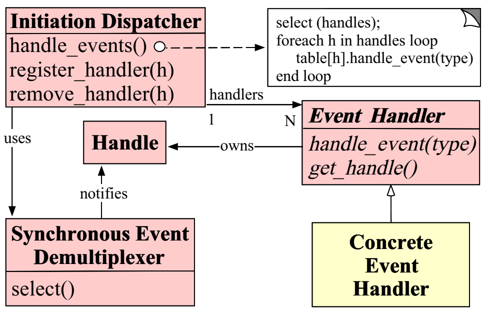
TAO

- TAO's wrapper facades are based on the ACE framework
- The Wrapper Facade pattern substantially increased portability and reduced the amount of *ad hoc* code

Addressing ORB Demuxing/Dispatching Challenges

- Problem
  - ORBs must process many different types of events simultaneously
- Forces
  - Multi-threading may not be available
  - Multi-threading may be inefficient
  - Multi-threading may be inconvenient
  - Tightly coupling general event processing with ORB-specific logic is inflexible
- Solution
  - Use the *Reactor* pattern to decouple generic event processing from ORB-specific processing

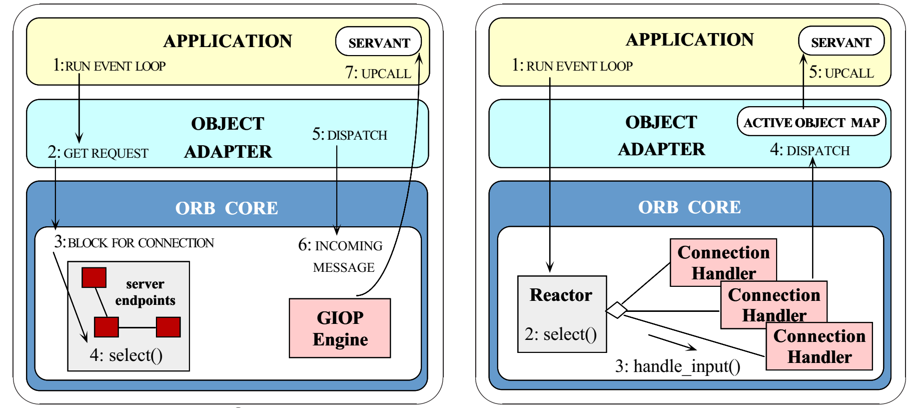
Enhancing Demuxing with the Reactor Pattern



www.cs.wustl.edu/$\sim$schmidt/Reactor.ps.gz

- **Intent**
  - *Decouples synchronous event demuxing/dispatching from event handling*
- **Forces Resolved**
  - Demuxing events efficiently within one thread
  - Extending applications without changing demux infrastructure

Using the Reactor Pattern in TAO



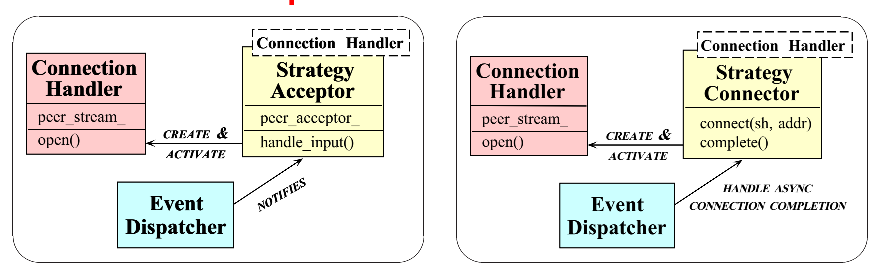
SunSoft IIOP **TAO**

- The Reactor pattern and ACE `Reactor` are widely used

Addressing ORB Endpoint Initialization Challenges

- Problem
  - The *communication* protocol used between ORBs is often orthogonal to its *connection establishment* and *service handler initialization* protocols
- Forces
  - Low-level connection APIs are error-prone and non-portable
  - Separating *initialization* from *processing* increases software reuse
- Solution
  - Use the *Acceptor-Connector* pattern to decouple passive/active connection establishment and GIOP connection handler initialization from the subsequent ORB interoperability protocol (*e.g.*, IIOP)

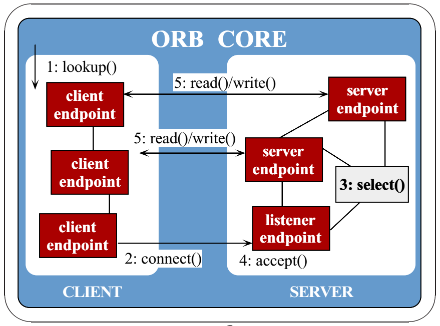
Enhancing Endpoint Initialization with the Acceptor-Connector Pattern



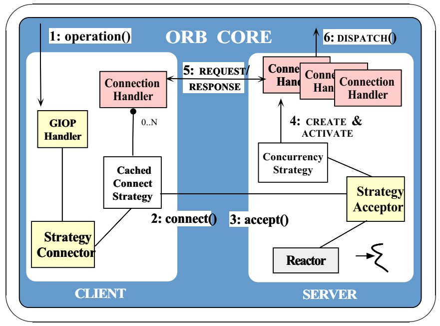
Acceptor Structure **Connector Structure**

- **Intent**
  - *Decouple connection establishment and service handler initialization from subsequent service processing*

Using the Acceptor-Connector Pattern in TAO



SunSoft IIOP



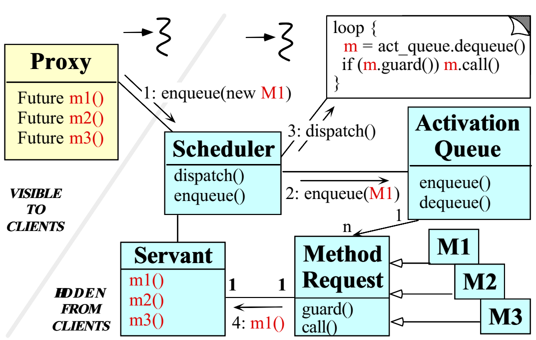
TAO

- **Forces Resolved**
  - (1) Improve portability and reuse and (2) avoid common mistakes

Addressing ORB Concurrency Challenges

- Problem
  - Multi-threaded ORBs are needed since Reactive ORBs are often inefficient, non-scalable, and non-robust
- Forces
  - Multi-threading can be very hard to program
  - No single multi-threading model is always optimal
- Solution
  - Use the *Active Object* pattern to allow multiple concurrent server operations using an OO programming style

Enhancing ORB Concurrency with the Active Object Pattern



www.cs.wustl.edu/∼schmidt/ Act-Obj.ps.gz

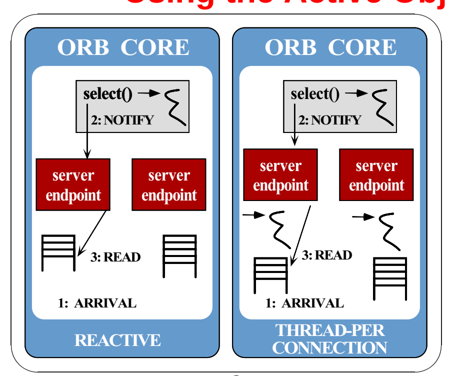
Intent

- *Decouple thread of request execution from thread of request reception*

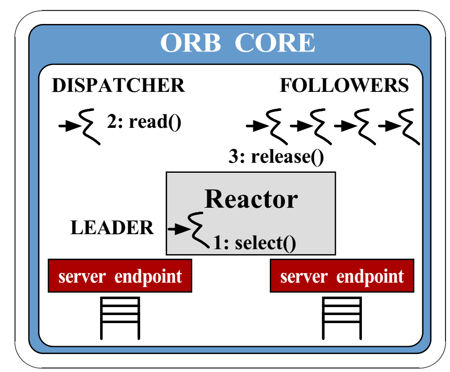
Forces Resolved

- Allow blocking operations
- Permit flexible concurrency strategies

Using the Active Object Pattern in TAO



SunSoft IIOP



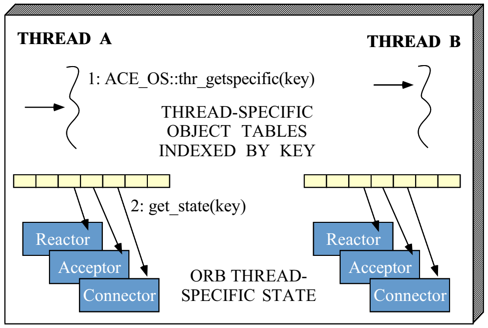
TAO

- TAO supports several variants of Active Objects (*e.g.*, Thread-per-Connection, Thread-per-Request, Thread Pool, etc.)

Reducing Lock Contention and Priority Inversions with the Thread-Specific Storage Pattern

- Problem
  - It is important to minimize the amount of locking required to serialize access to resources shared by an ORB
- Forces
  - Locks increase *performance overhead*
  - Locks increase potential for *priority inversion*
  - Different concurrency schemes yield different locking costs
- Solution
  - Use the *Thread-Specific Storage* pattern to maximize threading-model flexibility and minimize lock contention and priority inversion

Minimizing ORB Locking with the Thread-Specific Storage Pattern

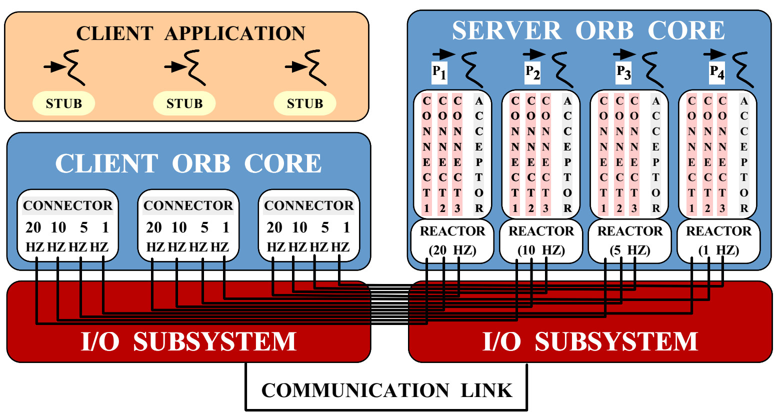


Intent

- *Allows multiple threads to use one logically global access point to retrieve ORB thread-specific data without incurring locking overhead for each access*

- **Forces Resolved**
  - Minimizes overhead and priority inversion

Using Thread-Specific Storage in TAO

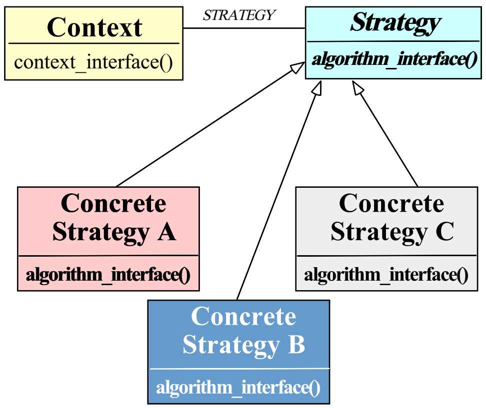


www.cs.wustl.edu/~schmidt/TSS-pattern.ps.gz

Addressing ORB Flexibility Challenges

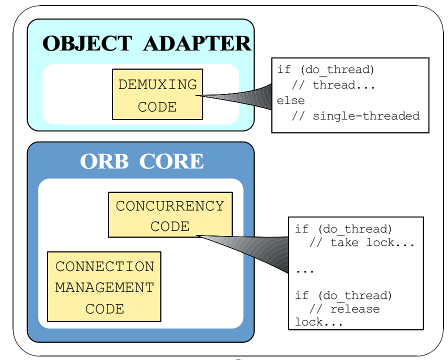
- Problem
  - Real-world ORBs must be flexible to satisfy the requirements of many different types of end-users and applications
- Forces
  - *Ad hoc* schemes for ORB flexibility are too static and non-extensible
  - Flexibility often has many (related) dimensions
- Solution
  - Use the *Strategy* pattern to support multiple transparently "pluggable" ORB strategies

Enhancing ORB Flexibility with the Strategy Pattern

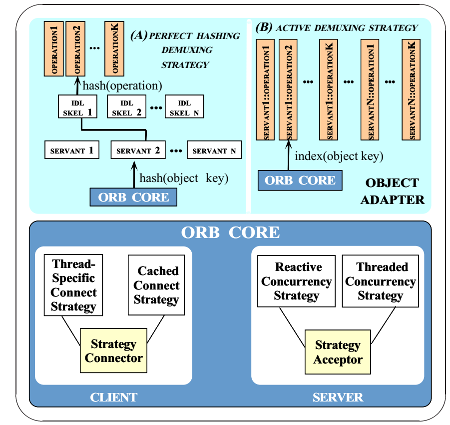


- **Intent**
  - *Factor out similarity among algorithmic alternatives*
- **Forces Resolved**
  - Orthogonally replace behavioral subsets transparently
  - Associating state with an algorithm

Using the Strategy Pattern in TAO



SunSoft IIOP

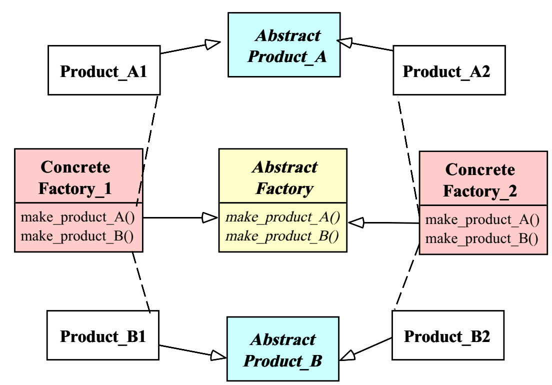


TAO

Addressing ORB Configurability Challenges

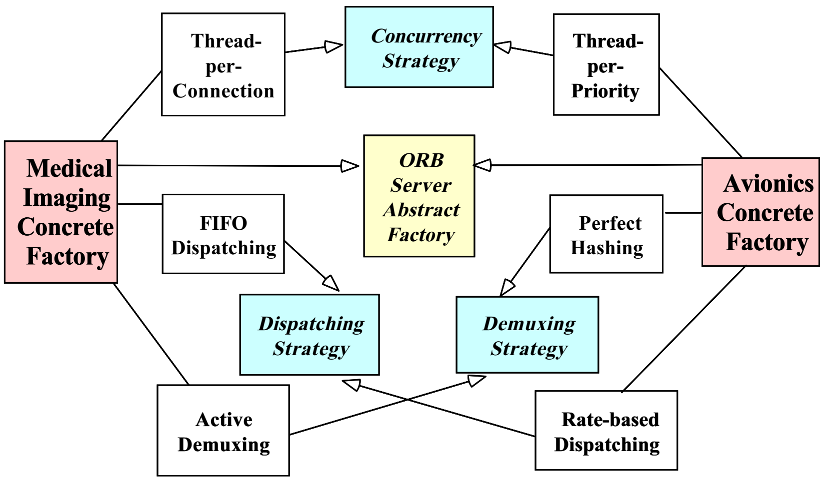
- Problem
  - Aggressive use of Strategy pattern creates a configuration nightmare
- Forces
  - Managing many individually configured strategies is hard
  - It's hard to ensure that groups of semantically compatible strategies are configured
- Solution
  - Use the *Abstract Factory* pattern to consolidate multiple ORB strategies into semantically compatible configurations

Centralizing ORB Configurability with the Abstract Factory Pattern



- **Intent**
  - *Integrate all strategies used to configure an ORB*
- **Forces Resolved**
  - Consolidates customization of many strategies
  - Ensures semantically-compatible strategies

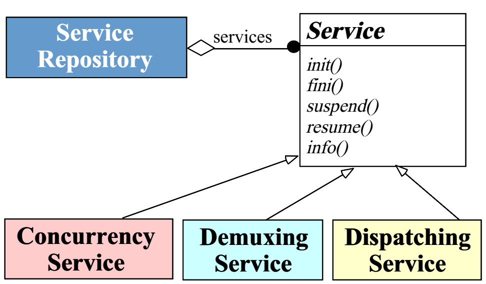
Using the Abstract Factory Pattern in TAO



Addressing ORB Dynamic Configurability Challenges

- Problem
  - Prematurely committing ourselves to a particular ORB configuration is inflexible and inefficient
- Forces
  - Certain ORB configuration decisions can't be made efficiently until run-time
  - Forcing users to pay for components they don't use is undesirable
- Solution
  - Use the *Service Configurator* pattern to assemble the desired ORB components dynamically

Enhancing Dynamic ORB Extensibility with the Service Configurator Pattern



www.cs.wustl.edu/∼schmidt/
Svc-Conf.ps.gz

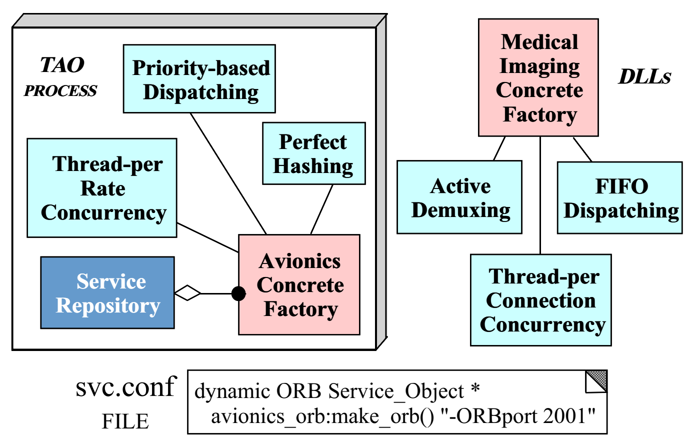
Intent

- *Decouples ORB strategies from time when they are configured*

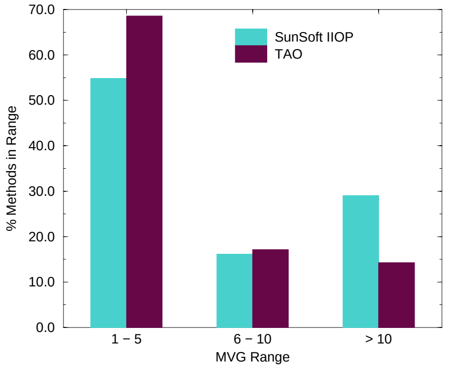
Forces Resolved

- Reduce resource utilization
- Support dynamic (re)configuration

Using the Service Configurator Pattern in TAO



Quantifying the Benefits of Patterns

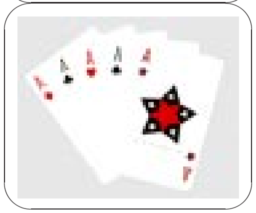


Macabe Complexity Metric Scores for TAO and SunSoft IIOP

- **Statistics**
  - Patterns greatly reduce code complexity
    - *e.g.*, Most TAO components have $v(G) < 10$
  - TAO components are substantially smaller than SunSoft IIOP
    - *e.g.*, connection management reduced by a factor of 5

Lessons Learned Developing QoS-enabled ORBs

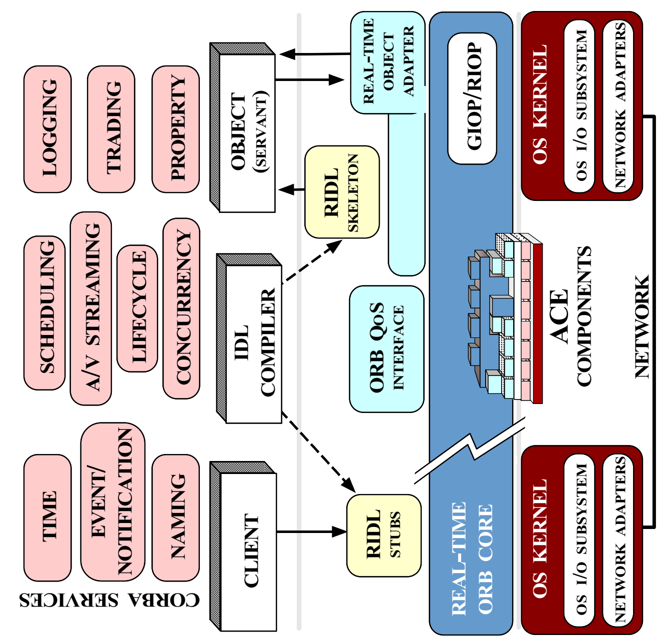
- Avoid dynamic connection management
- Minimize dynamic memory management and data copying
- Avoid multiplexing connections for different priority threads
- Avoid complex concurrency models
- Integrate ORB with OS and I/O subsystem and avoid reimplementing OS mechanisms
- Guide ORB design by empirical benchmarks and patterns



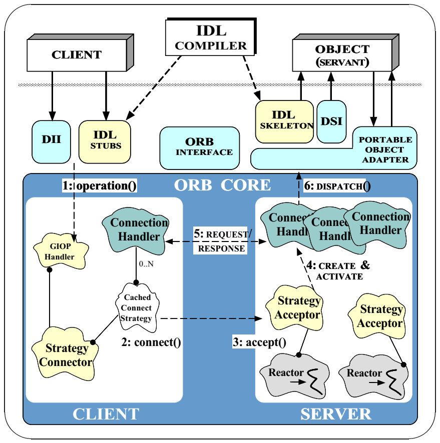
Concluding Remarks

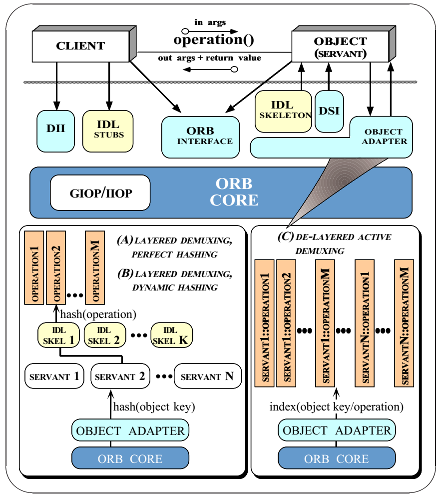
- Researchers and developers of distributed, real-time applications confront many common challenges
  - *e.g.*, service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful researchers and developers apply *patterns*, *frameworks*, and *components* to resolve these challenges
- Careful application of patterns can yield efficient, predictable, scalable, *and* flexible middleware
  - *i.e.*, middleware performance is largely an “implementation detail”
- Next-generation ORBs will be highly QoS-enabled, though many research challenges remain

Current Status of TAO



Synopsis of TAO's Pattern-Oriented ORB Design





Summary of TAO Research Project

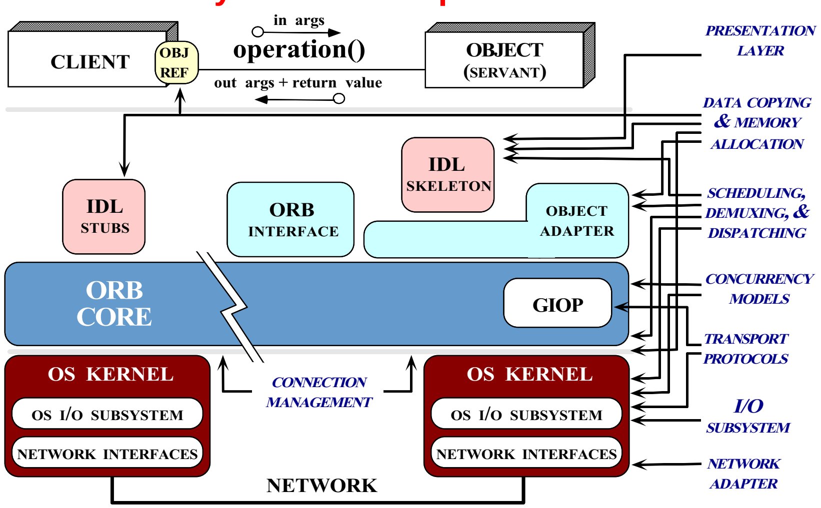
Completed work

- First POA and first deployed real-time CORBA scheduling service
- Pluggable protocols framework
- Minimized ORB Core priority inversion and non-determinism
- Reduced latency via demuxing optimizations
- Co-submitters on OMG's real-time CORBA spec

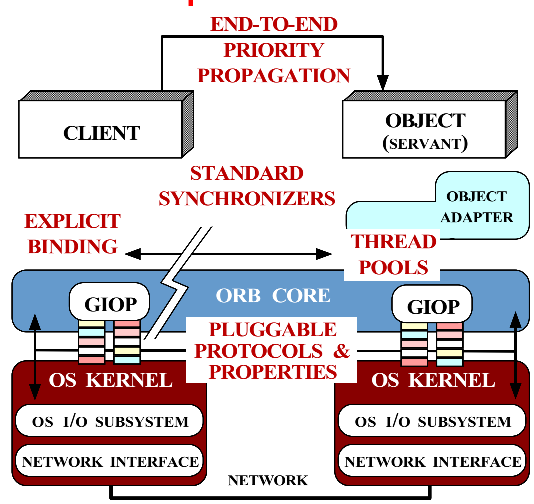
Ongoing work

- Dynamic/hybrid scheduling
- Distributed QoS, ATM I/O Subsystem, & open signaling
- Implement CORBA Real-time, Messaging, and Fault Tolerance specs
- Tech. transfer via DARPA Quorum program and www.theaceorb.com
  - Integration with Flick IDL compiler, QuO, TMO, etc.

Summary: Real-time Optimizations in TAO



Next Steps: New TAO Features and Optimizations

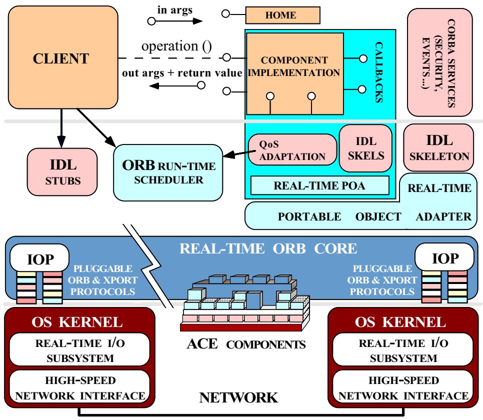


Forthcoming Features

- CORBA Component Model (CCM)
- Real-time and Minimum CORBA
- CORBA Messaging
- Fault-Tolerant CORBA
- Notification Service

www.cs.wustl.edu/~schmidt/TAO-status.html

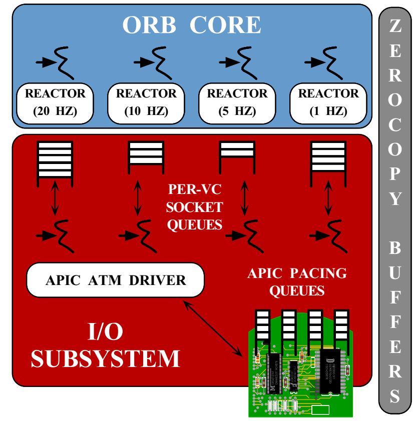
Next Steps: Integrating QoS-Enabled CORBA Component Model with TAO



Features

- Select optimal communication reflectively
- Re-factor component QoS aspects into their containers
- Dynamically load/unload component implementations

Next Steps: Integrating TAO with ATM I/O Subsystem

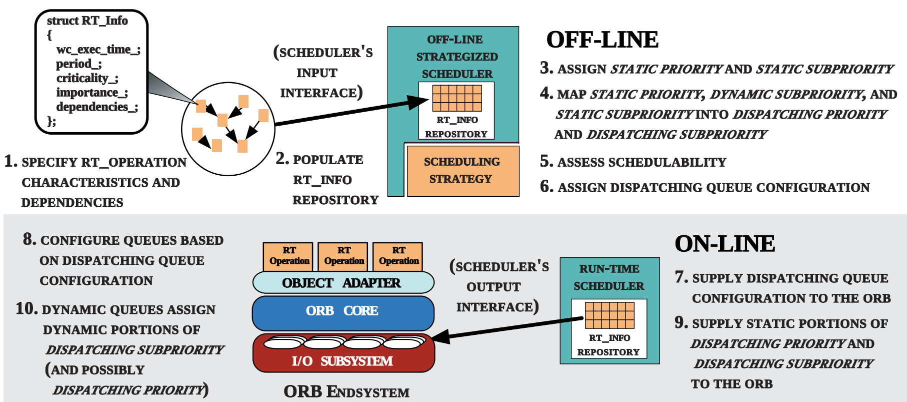


~schmidt/RIO.ps.gz

Features

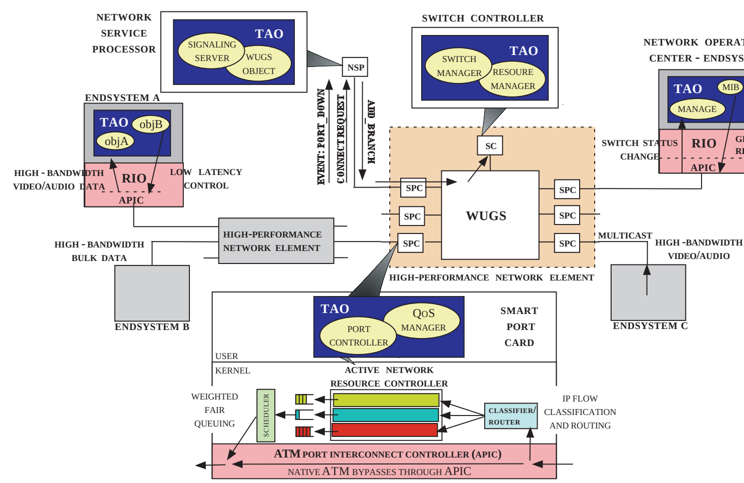
- Vertical integration of QoS through ORB, OS, and ATM network
- Real-time I/O enhancements to Solaris kernel
- Provides rate-based QoS end-to-end
- Leverages APIC features for cell pacing and zero-copy buffering

Next Steps: Strategized Scheduling Framework

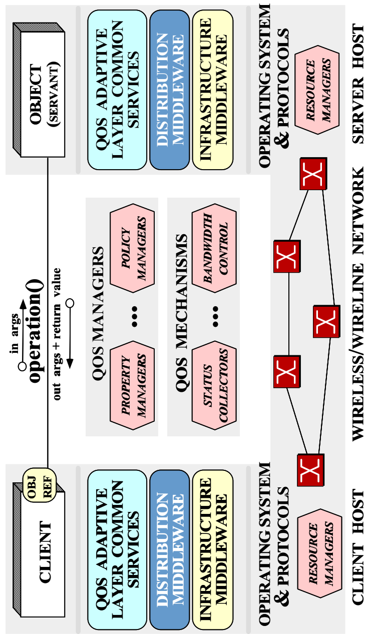


www.cs.wustl.edu/~schmidt/dynamic.ps.gz

Next Steps: Open ATM Signaling & Control



Next Steps: Adaptive Middleware (e.g., QuO/TAO)

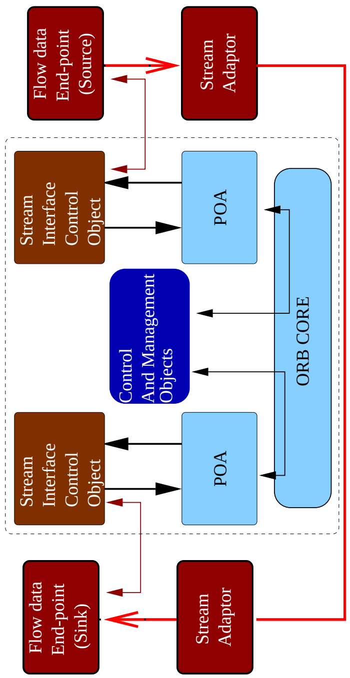


www.dist-systems.bbn.com/papers/

Key Themes

- Decouple *functional* path from *QoS* path
- Emphasize integration and configuration

Next Steps: Audio/Video Streaming



www.cs.wustl.edu/~schmidt/av.ps.gz

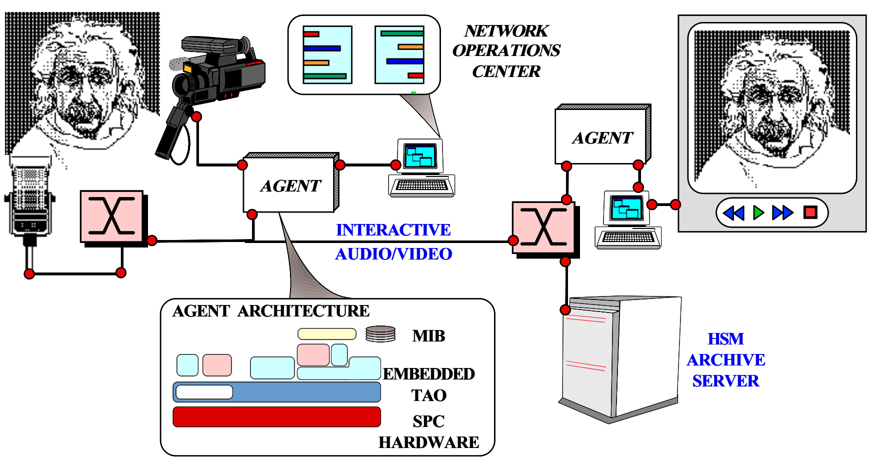
Efficiency

- Sockets for data transfer to get high performance

Flexibility

- Uses CORBA for control messages and properties

Next Steps: Distributed Interactive Simulations



Web URLs for Additional Information

- These slides: `~schmidt/TAO4.ps.gz`
- More information on CORBA: `~schmidt/corba.html`
- More info on ACE: `~schmidt/ACE.html`
- More info on TAO: `~schmidt/TAO.html`
- TAO Event Channel: `~schmidt/JSAC-98.ps.gz`
- TAO static scheduling: `~schmidt/TAO.ps.gz`
- TAO dynamic scheduling: `~schmidt/dynamic.ps.gz`
- ORB Endsystem Architecture: `~schmidt/RIO.ps.gz`
- Pluggable protocols: `~schmidt/pluggable_protocols.ps.gz`

Web URLs for Additional Information (cont'd)

- Network monitoring, visualization, & control: `~schmidt/NMVC.html`
- Performance Measurements:
  - Demuxing latency: `~schmidt/COOTS-99.ps.gz`
  - SII throughput: `~schmidt/SIGCOMM-96.ps.gz`
  - DII throughput: `~schmidt/GLOBECOM-96.ps.gz`
  - ORB latency & scalability: `~schmidt/ieee_tc-97.ps.gz`
  - IIOP optimizations: `~schmidt/JSAC-99.ps.gz`
  - Concurrency and connection models: `~schmidt/RT-perf.ps.gz`
  - RTOS/ORB benchmarks:
    `~schmidt/RT-OS.ps.gz`
    `~schmidt/words-99.ps.gz`