QoS-enabled Middleware for Managing and Controlling High-Speed Networks

Douglas C. Schmidt

Associate Professor &
Director of the Center for
Distributed Object computing

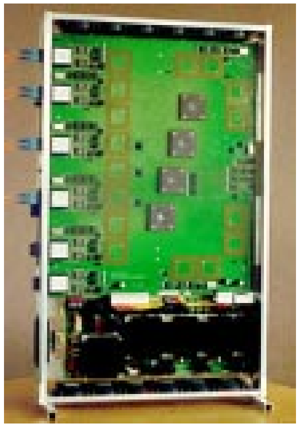
Computer Science Dept.
Washington University, St. Louis
www.cs.wustl.edu/∼schmidt/

Sponsors

NSF, DARPA, Bellcore/Telcordia, BBN, Boeing, CDI/GDIS, Hughes, Kodak, Lockheed, Lucent, Microsoft, Motorola, Nokia, Nortel, OCI, OTI, SAIC, Siemens SCR, Siemens MED, Siemens ZT, Sprint, USENIX

April 13th, 1999

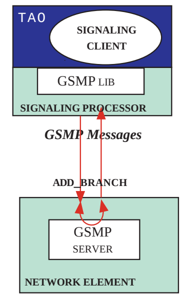
Motivation: the High-speed Network Software Crisis



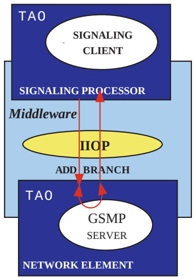
www.arl.wustl.edu/arl/

- **Symptoms**
  - Network element ***hardware*** gets smaller, faster, cheaper
  - Control/management ***software*** gets larger, slower, more expensive
- **Culprits**
  - ***Inherent*** and ***accidental*** complexity
- **Solution Approach**
  - Manage & control network elements via QoS-enabled embedded middleware

General Switch Management Protocol (GSMP)



Conventional Implementation



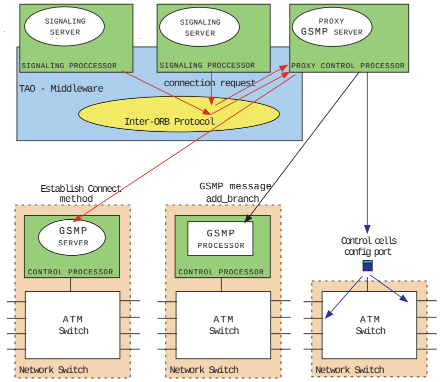
CORBA Based Implementation

Features

- Setup & release connections
- Add & delete point-to-multipoint leaves
- Manage ATM switch ports
- Request configuration information & statistics

Low-level C APIs send RFC 2297 GSMP ATM messages

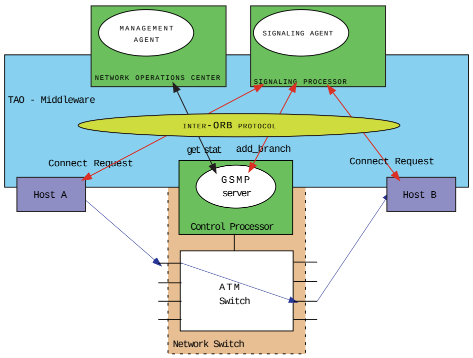
GSMP Proxy Server Configuration



Features

- Supports standard CORBA programming API
- Can use standard ORB
- Transparent to existing GSMP servers
- Scales to distributed configuration
  - *i.e.*, one CP can control multiple switches

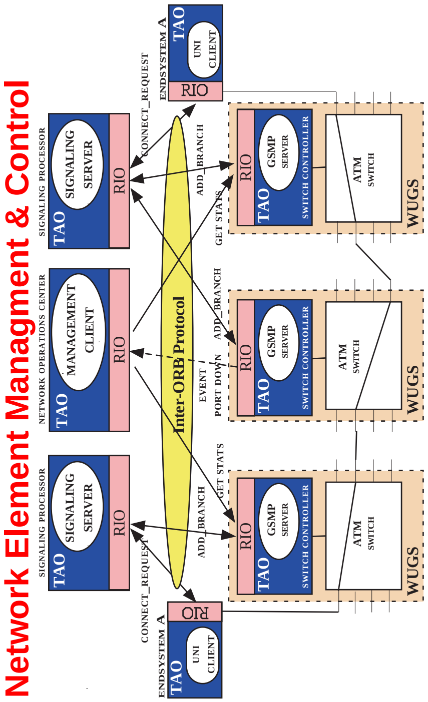
GSMP Embedded ORB Configuration



Features

- Leverages middleware flexibility and standardization
- Multiple protocols can be supported
  - GSMP in-line bridging, IIOP, etc.

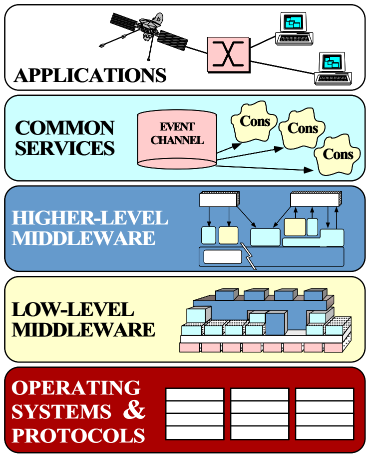
Goal: Apply Embedded Middleware to Network Element Managment & Control



Domain Challenges

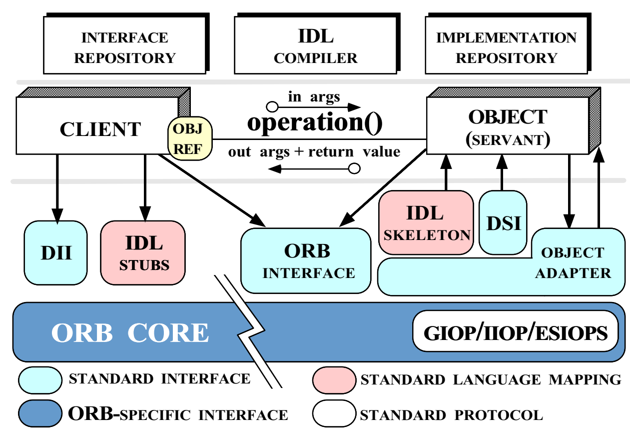
- High-speed (20 Gbps) ATM switches w/embedded controllers
- Low-latency and statistical real-time deadlines
- COTS infrastructure, open systems, and small footprint

Problem: Lack of QoS-enabled Middleware



- Many telecom applications require QoS guarantees
  - *e.g.*, call-processing, network/switch management, wireless
- Building these applications manually is hard
- Existing middleware doesn't support QoS effectively
  - *e.g.*, CORBA, DCOM, DCE, Java
- Solutions must be integrated horizontally & vertically

Candidate Solution: CORBA

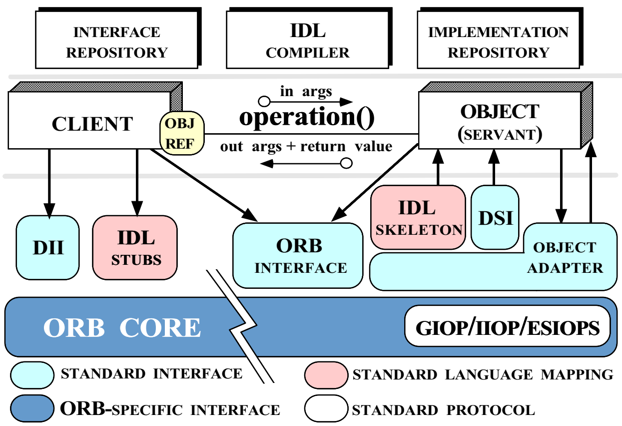


www.cs.wustl.edu/∼schmidt/corba.html

Goals of CORBA

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

Caveat: Limitations of CORBA for QoS

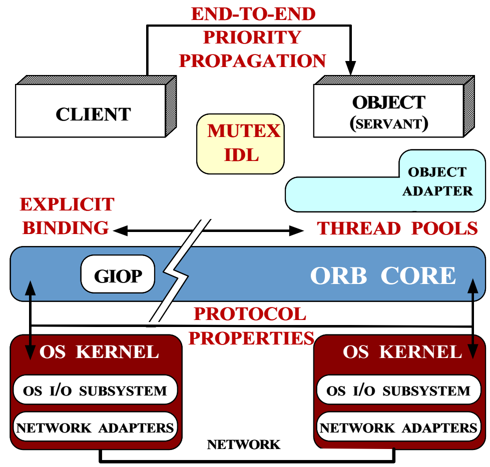


www.cs.wustl.edu/∼schmidt/RT-ORB.ps.gz

Limitations

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

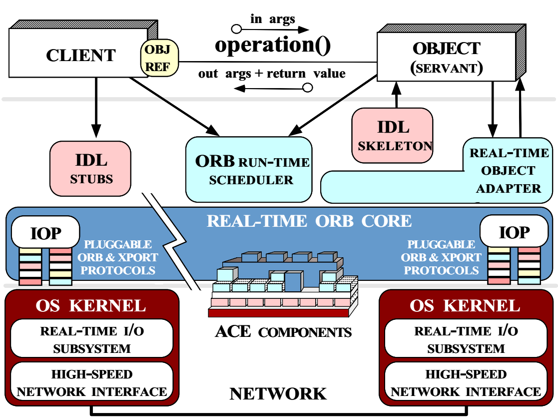
Overview of the Real-time CORBA Specification



Features

1. End-to-end priority propagation
2. Protocol properties
3. Thread pools
4. Explicit binding
5. Mutex IDL

Our Approach: The ACE ORB (TAO)

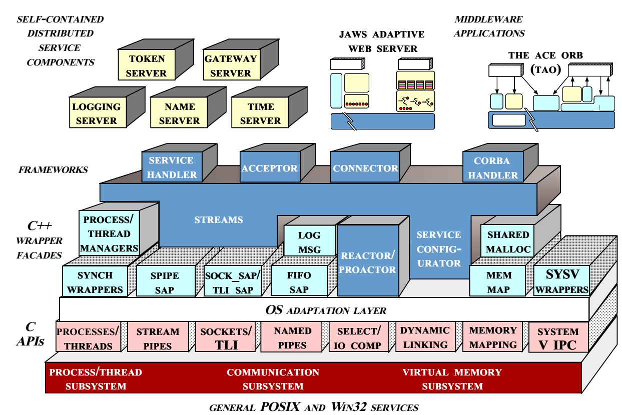


www.cs.wustl.edu/∼schmidt/TAO.html

TAO Overview →

- An open-source, standards-based, real-time, high-performance CORBA ORB
- Runs on POSIX, Win32, & embedded RT 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

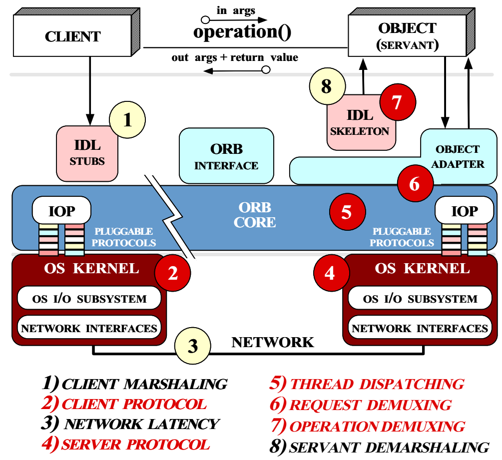
Related work →

- x-Kernel
- SysV STREAMS

ACE and TAO Statistics

- Over 30 person-years of effort
  - ACE > 185,000 LOC
  - TAO > 100,000 LOC
  - TAO IDL compiler > 100,000 LOC
  - TAO CORBA Object Services > 150,000 LOC
- Ported to UNIX, Win32, MVS, and RTOS platforms
- Large user community
  - www.cs.wustl.edu/∼schmidt/ACE-users.html
- Currently used by dozens of companies
  - Bellcore, Boeing, Ericsson, Kodak, Lockheed, Lucent, Motorola, Nokia, Nortel, Raytheon, SAIC, Siemens, etc.
- Supported commercially
  - ACE → www.riverace.com
  - TAO → www.ociweb.com

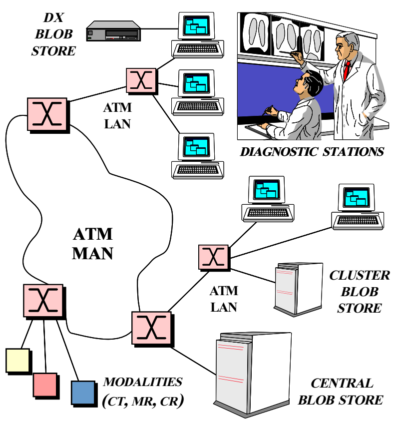
Optimization Challenges for QoS-enabled ORBs



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

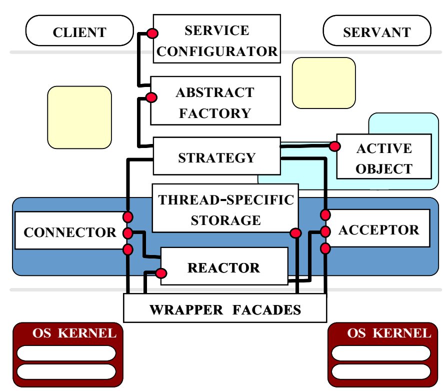
Common Problems →

- 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: 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

ORB Optimization Principle Patterns

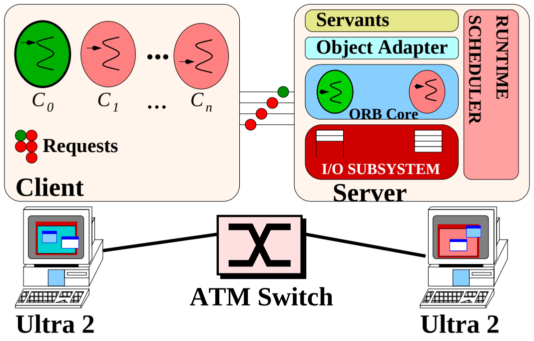
Definition

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

Key Principle Patterns Used in TAO

#	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

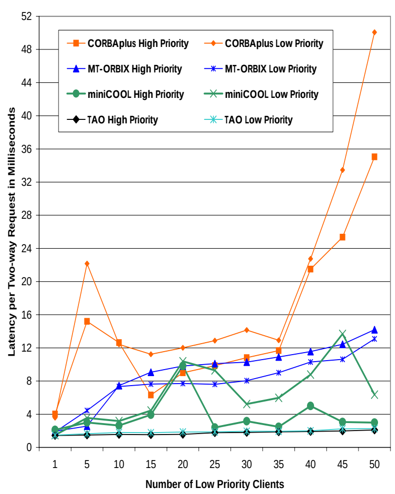


www.cs.wustl.edu/∼schmidt/RT-perf.ps.gz

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 twoway CORBA calls that cube a number on the servant and returns result

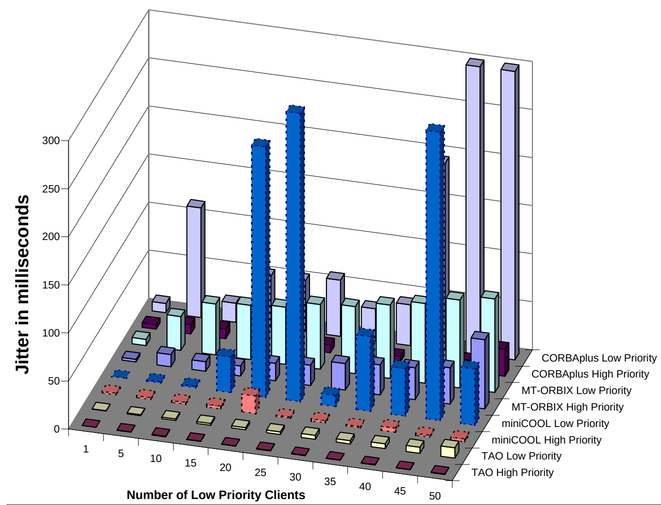
ORB Latency and Priority Inversion Results



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



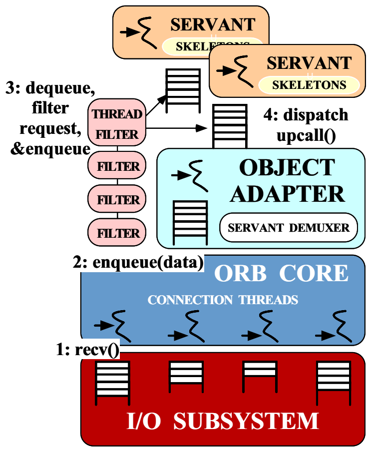
Definition

- 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

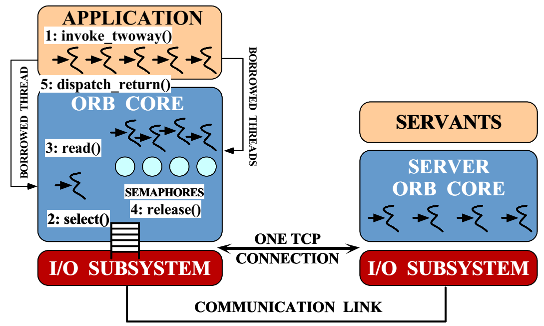


Common Problems

- 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

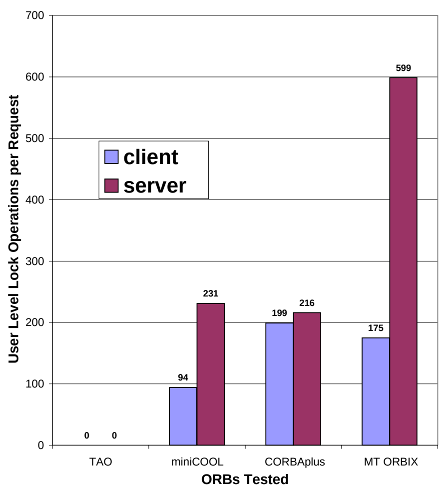


Common Problems

- 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

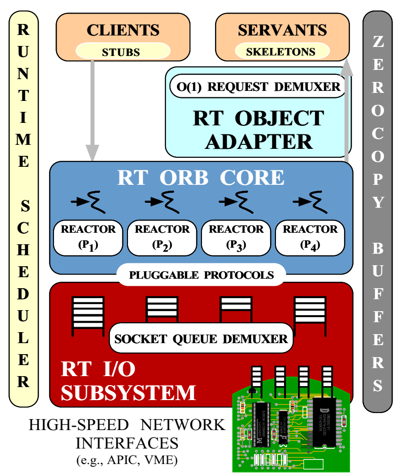


Common Problems

- 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



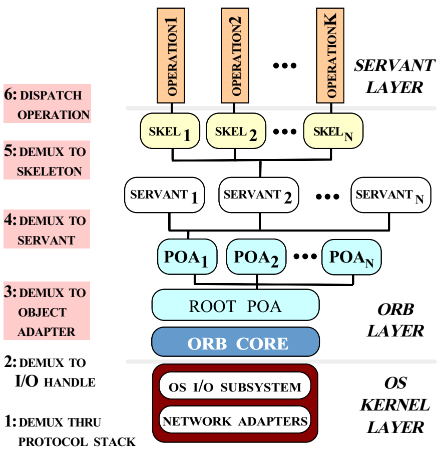
Solution Approach →

- Integrate scheduler into ORB endsystem
- Support multiple scheduling strategies
- Co-schedule threads

Principle Patterns →

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

Problem: Reducing Demultiplexing Latency

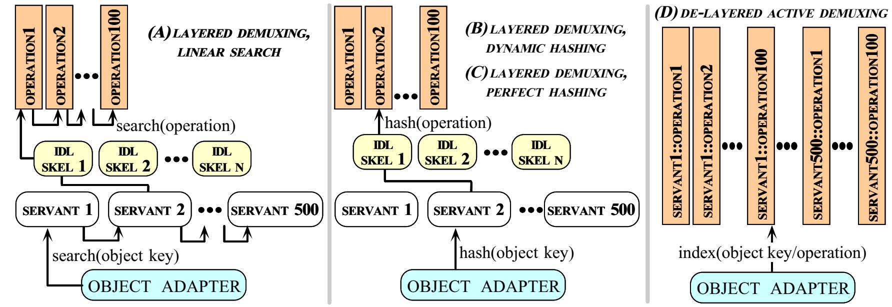


Design Challenges

- 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

Solution: TAO's Request Demultiplexing Optimizations



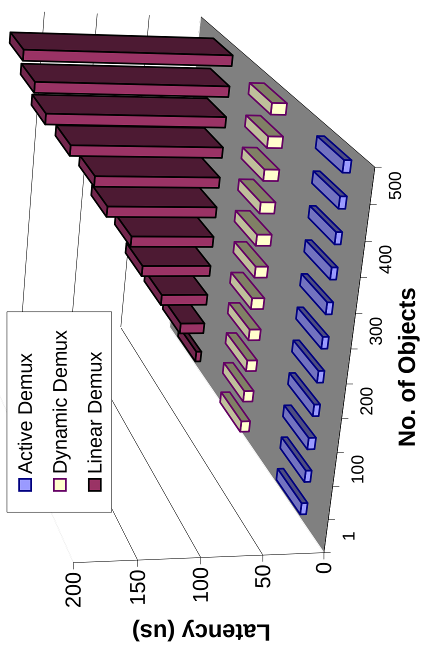
Demuxing

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

Perfect hashing

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

Servant Demultiplexing Results



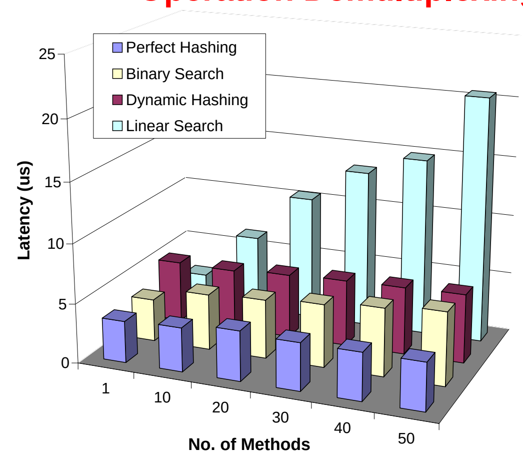
Synopsis of Results

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

Principles

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

Operation Demultiplexing Results



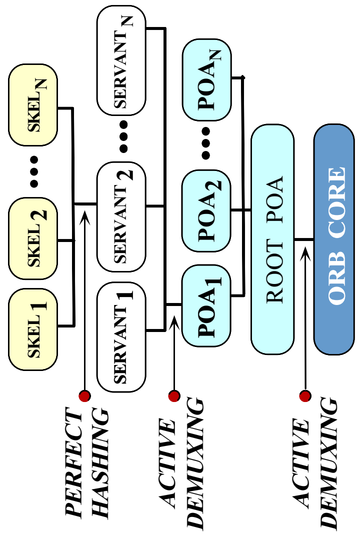
Synopsis of Results →

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

Principle Patterns →

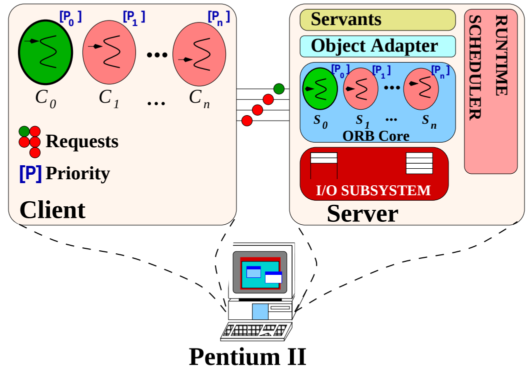
- Precompute, use predictable data structures, remove gratuitous waste

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

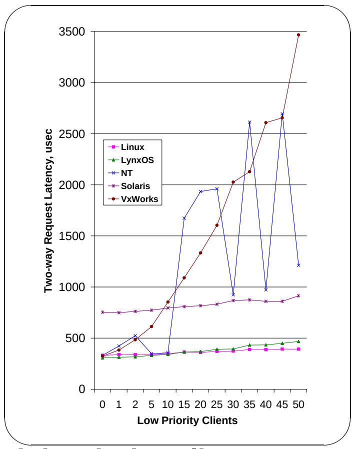


www.cs.wustl.edu/~schmidt/RT-OS.ps.gz

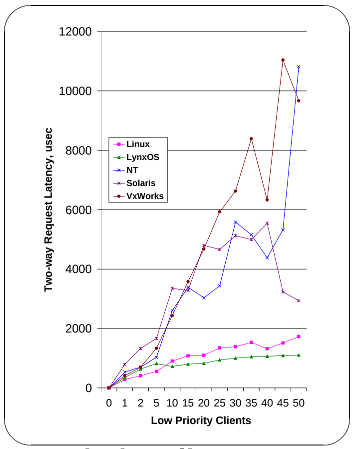
Method

- 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 twoway CORBA calls that cube a number on the servant and returns result

Real-time ORB/OS Performance Results

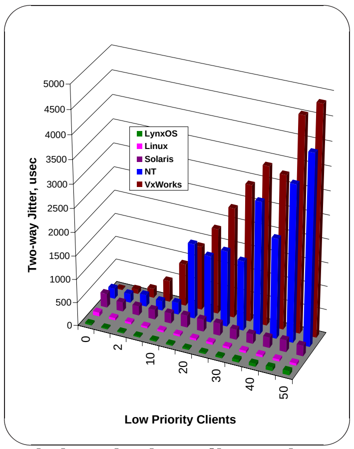


High-priority Client Latency

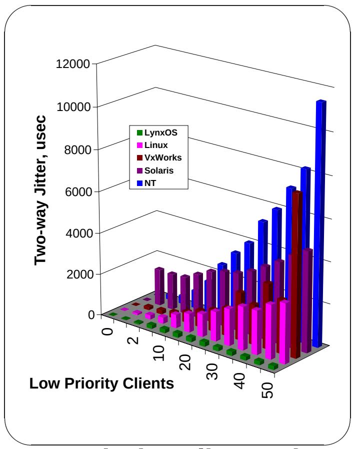


Low-priority Clients Latency

Real-time ORB/OS Jitter Results

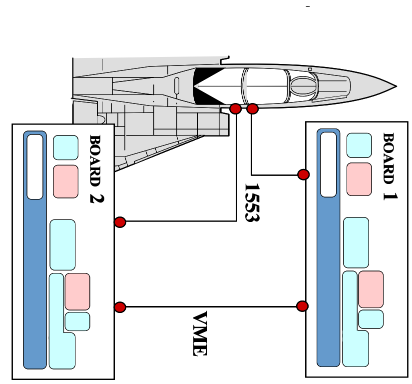


High-priority Client Jitter



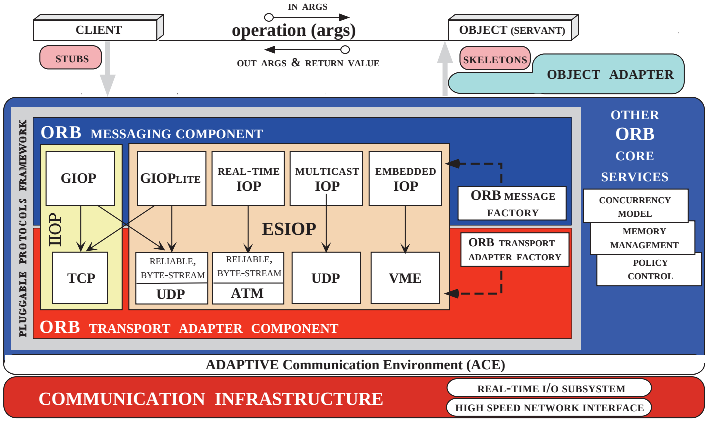
Low-priority Clients Jitter

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
- Existing ORBs don't support "pluggable protocols"
  - This makes ORBs inflexible and inefficient

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

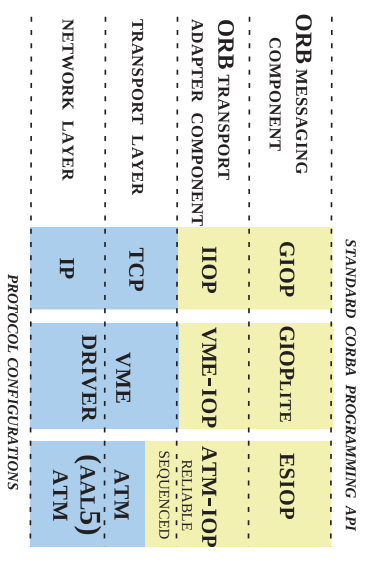
One Solution: Hacking GIOP

- GIOP requests include fields that aren't needed in homogeneous embedded applications
  - *e.g.*, GIOP magic #, GIOP version, byte order, request principal, etc.
- TAO's `gioplite` 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 measureable, but small (2%), improvement in throughput/latency
- Our pluggable protocols framework will all much greater decreases in IOP request sizes, as well as more flexible support for multiple transport protocols
- However, there will be no changes required to the standard CORBA programming model

CORBA Protocol Interoperability Architecture

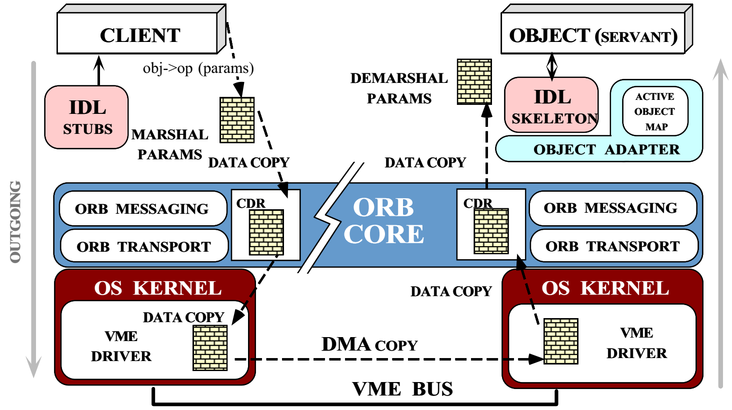


Features →

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

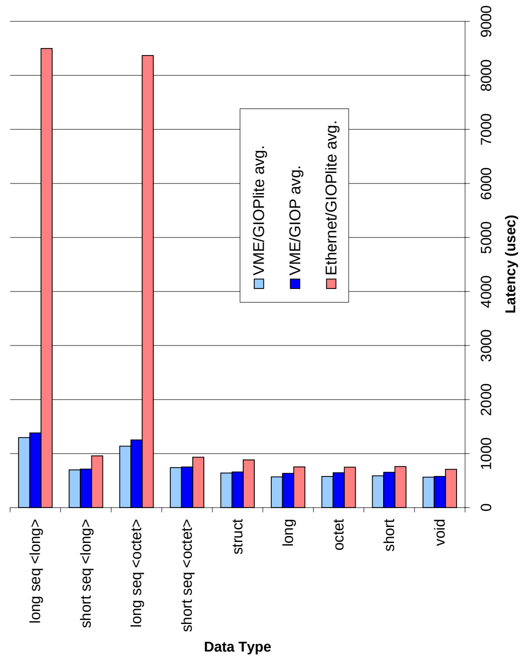
www.cs.wustl.edu/∼schmidt/ pluggable-protocols.ps.gz

Embedded System Benchmark Configuration



VxWorks running on PowerPC over 320 Mbps VME & Ethernet

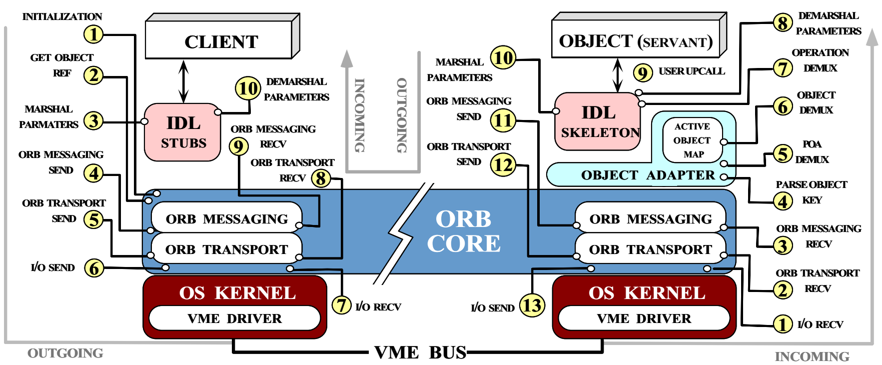
Ethernet & VME Two-way Latency Results



Synopsis of Results

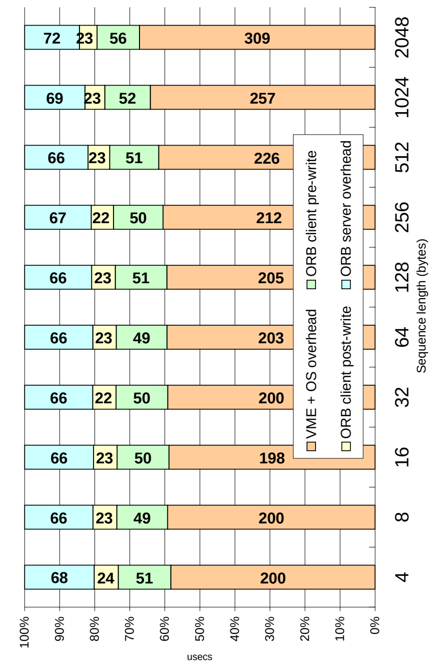
- VME protocol is much faster than Ethernet
- No application changes are required to support VME

Pinpointing ORB Overhead with VMEtro Timeprobes



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

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

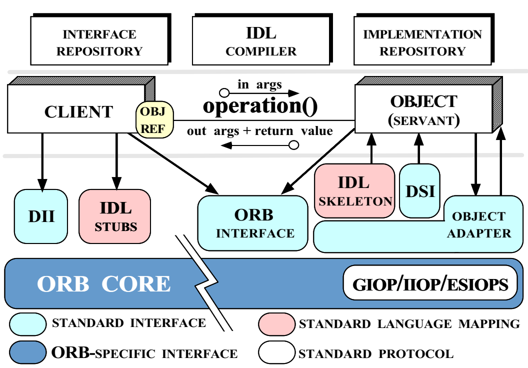
Client Whitebox Latency Results

Direction	Client Activities	Absolute Time (μs)
Outgoing	1. *Initialization*	36
	2. *Get object reference*	12
	3. *Parameter marshal*	operation dependent
	4. *ORB messaging send*	4
	5. *ORB transport send*	2
	6. *I/O send*	operation dependent
Incoming	7. *I/O receive*	operation dependent
	8. *ORB transport recv*	2
	9. *ORB messaging recv*	12
	10. *Parameter demarshal*	operation dependent

Server Whitebox Latency Results

Direction	Server Activities	Absolute Time (μs)
Incoming	1. *I/O receive*	operation dependent
	2. *ORB transport recv*	10
	3. *ORB messaging recv*	33
	4. *Parsing object key*	12
	5. *POA demux*	3
	6. *Servant demux*	6
	7. *Operation demux*	4
	8. *Parameter demarshal*	operation dependent
	9. *User upcall*	servant dependent
Outgoing	10. *Return value marshal*	operation dependent
	11. *ORB messaging send*	34
	12. *ORB transport send*	3
	13. *I/O send*	operation dependent

Problem: Overly Large Memory Footprint



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

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

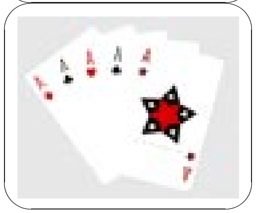
Solution: Minimum CORBA

Component	CORBA	Minimum CORBA	Percentage Reduction
POA	281,896	207216	26.5
ORB Core	347,080	330,304	4.8
Dynamic Any	131,305	0	100
CDR Interpreter	68,687	68,775	0
IDL Compiler	10,488	10,512	0
Pluggable Protocols	14,610	14,674	0
Default Resources	7,919	7,975	0
Total	861,985	639,456	25.8

Applying Minimum CORBA subsetting to TAO reduces memory footprint by ~25% and increases ORB determinism

Lessons Learned Developing QoS-enabled ORBs

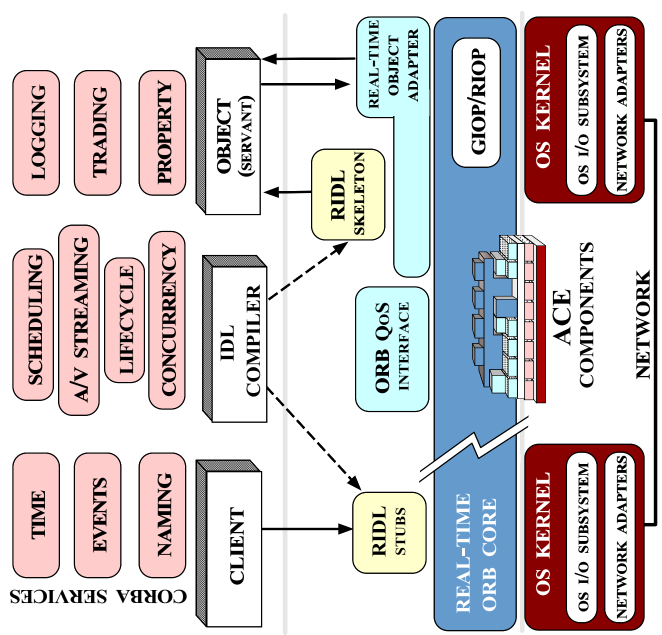
- Avoid dynamic connection management
- Minimize dynamic memory management and data copying
- Avoid multiplexing connections for different priority threads
- Avoid complex concurrency models
- Integrate ORB with OS and I/O subsystem and avoid reimplementing OS mechanisms
- Guide ORB design by empirical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



Concluding Remarks

- Researchers and developers of distributed, real-time, embedded telecom applications confront 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 for telecom will be highly QoS-enabled, though many research challenges remain

Current Status of TAO



Summary of TAO Research Project

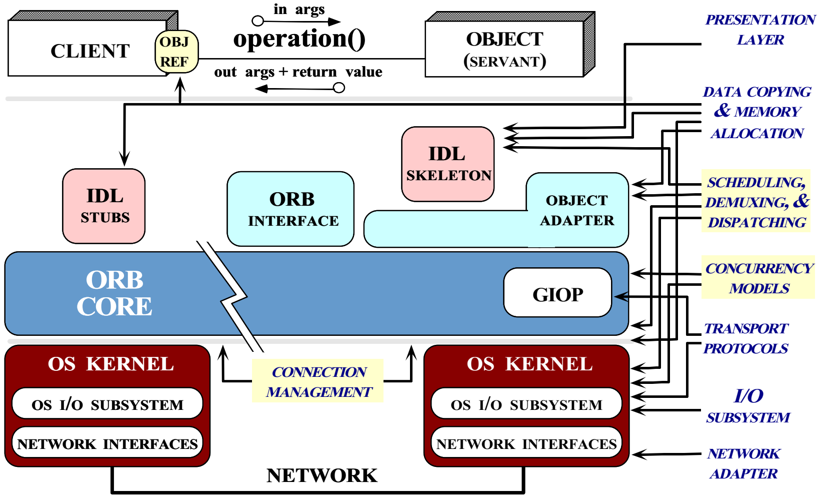
Completed work

- 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

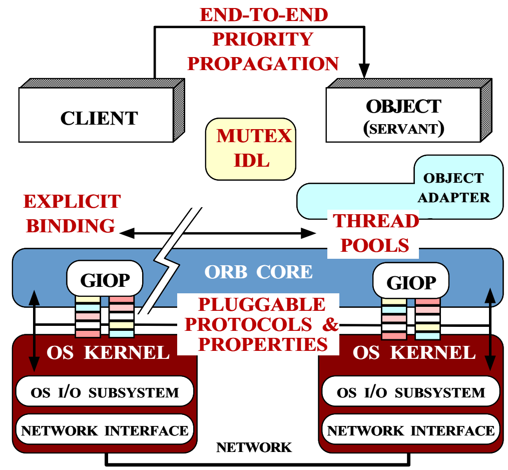
Ongoing work

- Dynamic/hybrid scheduling of CORBA operations
- Distributed QoS, ATM I/O Subsystem, & open signaling
- Implement Real-time CORBA spec
- Tech. transfer via DARPA Quorum program and www.ociweb.com

Summary: Real-time Optimizations in TAO



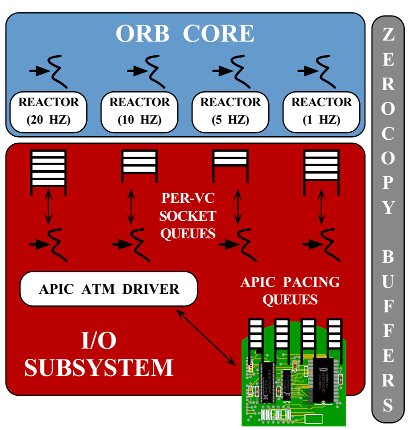
Next Steps: New TAO Features and Optimizations



- **New Features**
  - Real-time CORBA
  - Minimum CORBA
  - CORBA Messaging
  - Fault tolerance
- **New Optimizations**
  - Startup time
  - Event Channel
  - POA hierarchies

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

Next Steps: Integrating TAO with ATM I/O Subsystem

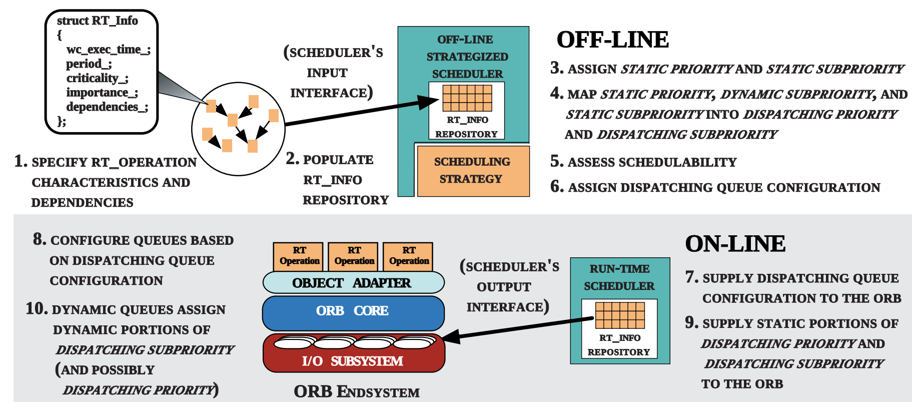


~schmidt/RIO.ps.gz

Key Features

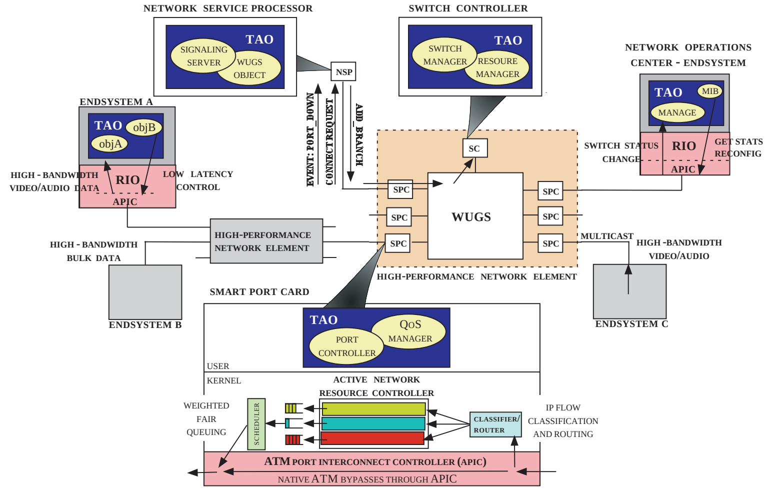
- Vertical integration of QoS through ORB, OS, and ATM network
- Real-time I/O enhancements to Solaris kernel
- Provides rate-based QoS end-to-end
- Leverages APIC features for cell pacing and zero-copy buffering

Next Steps: Strategized Scheduling Framework

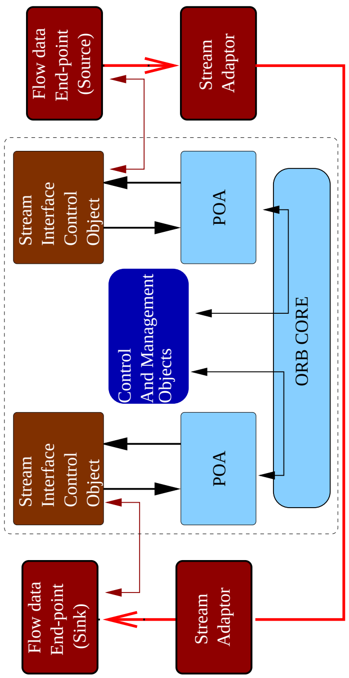


www.cs.wustl.edu/~schmidt/dynamic.ps.gz

Next Steps: Open ATM Signaling & Control



Next Steps: 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

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)

- 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`