WASHINGTON STATE UNIVERSITY

School of Electrical Engineering
and Computer Science

MicroQoSCORBA

A QoS-Enabled, Reflective, and Configurable Middleware Framework for Embedded Systems

Professors: **David E. Bakken**, John C. Shovic

Students: A. David McKinnon, Olav Haugan*, Tarana Damania, Kevin Dorrow, Dr. Wesley Lawrence

*Former Student

June 2002

© 2002 David E. Bakken

My Background

- **WSU '85** (BS CptS, BS Math, Minor EE)
- Boeing (Seattle) 1985-1988
  - Flight Simulation Lab & Data_Flow tool to parallelize legacy flight simulation code
  - 7J7 Real-Time Executive R&D
  - Taught Ada, C, Unix
- Grad school at U. Arizona, Tucson 1988-1994
  - distributed fault tolerance, some "portable parallelism"
- BBN (Boston) 1994-1999 (BBN built the first Internet in 1969 for DARPA…)
  - Co-architect of **Quality Objects (QuO) framework** for QoS for middleware (CORBA initially), now 7 DARPA projects >$10 million
  - PI on DARPA AQuA project to add replication support to CORBA (with Cornell and U. Illinois at Urbana-Champaign)
  - Replication consultant to BBN/GTE-Internetworking national ISP
- **WSU** June 1999 until retirement…. ☺
  - Consult to Trusted Information Systems (now "NAI Labs"), TriGeo Net. Security

© 2002 David E. Bakken

Outline

- **Embedded systems market background**
- Middleware and its benefits
- MicroQoSCORBA Baseline
  - Lifecycle Time Epochs
    - Middleware Taxonomy
  - Base Architecture
  - CASE Tools
  - Profiling (work in progress; done Aug. ’02)
  - Implementation
- MicroQoSCORBA Quality of Service (QoS)
  - Fault Tolerance (done July ‘02)
  - RealTime (Work in Progress; done Dec. ‘02)
  - Security (Work in Progress; done Dec ‘02)
  - Possible Boeing-MicroQoSCORBA interactions
- Key Research issues (won’t get to all today….)

© 2002 David E. Bakken

Embedded Systems Market

- System size varies
  - Aircraft
  - PDAs
  - Home appliances
- 11 Billion parts per year
- Application volume varies
  - Automobiles, TVs
  - Satellites

© 2002 David E. Bakken

The Network is Coming

- Embedding the Internet, CACM, May 2000
  - Smart building materials, wireless computing, smart pills, dog collars, street lights, …
- Wireless computing
- Sensor Networks
  - Environmental monitoring, Home alarm system
  - Wildlife research
- Disposable/One-Time Use
  - Hazardous environments, Battlefield deployment
  - Or simply inexpensive enough to leave behind!

© 2002 David E. Bakken

Outline

- Embedded systems market background
- **Middleware and its benefits**
- MicroQoSCORBA Baseline
  - Lifecycle Time Epochs
    - Middleware Taxonomy
  - Base Architecture
  - CASE Tools
  - Profiling (work in progress; done Aug. ’02)
  - Implementation
- MicroQoSCORBA Quality of Service (QoS)
  - Fault Tolerance (done July ‘02)
  - RealTime (Work in Progress; done Dec. ‘02)
  - Security (Work in Progress; done Dec ‘02)
  - Possible Boeing-MicroQoSCORBA interactions
- Key Research issues (won’t get to all today….)

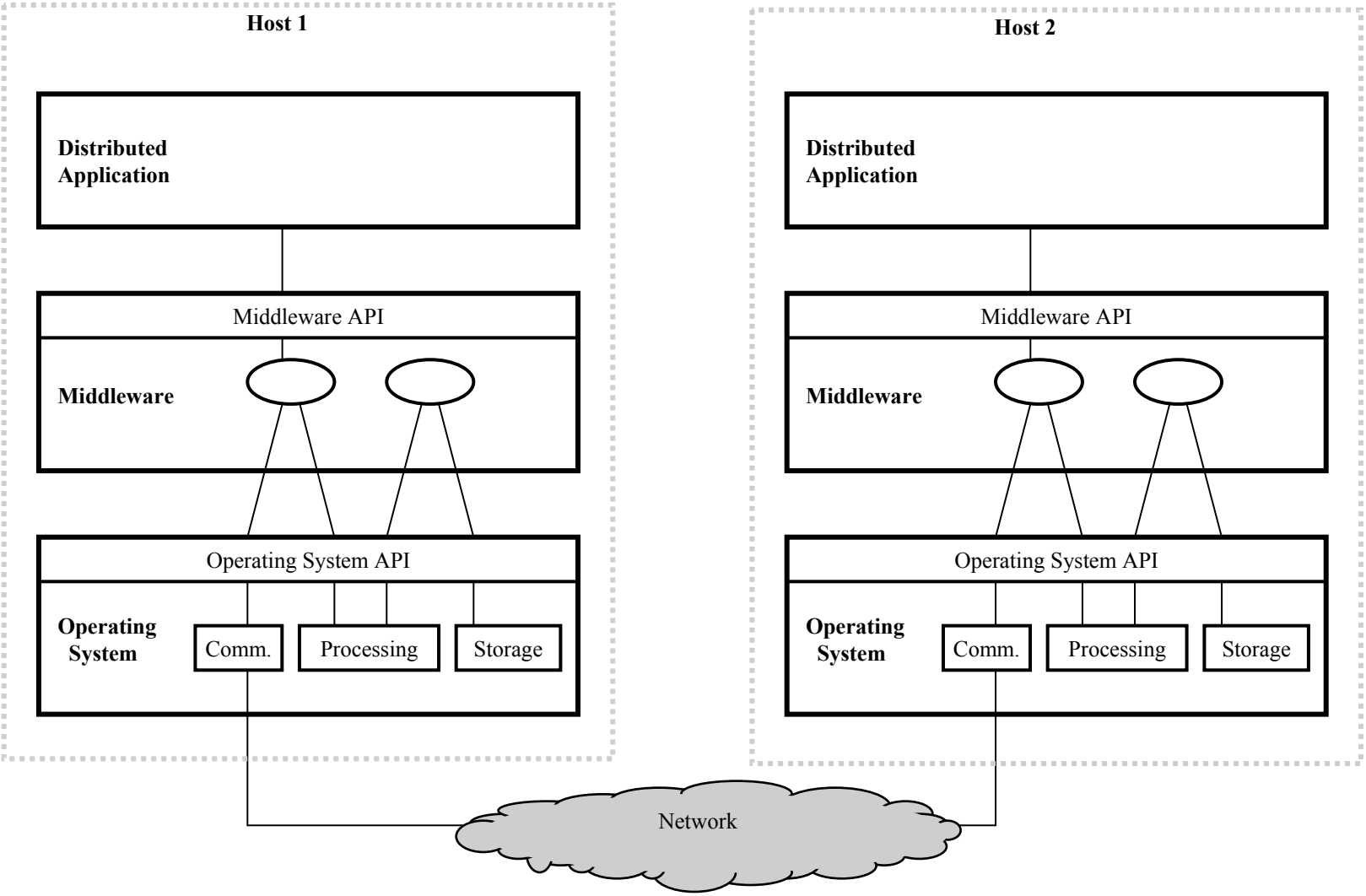
© 2002 David E. Bakken

Why Middleware?

- Middleware == "**A layer of software above the operating system but below the application program that provides a common programming abstraction across a distributed system**"
- Middleware exists to help manage the complexity and heterogeneity inherent in distributed systems
- Middleware provides higher-level building blocks ("abstractions") for programmers than the OS provides
  - Can make code much more portable
  - Can make them much more productive
  - Can make the resulting code have fewer errors
- Middleware sometimes is called "plumbing"
  - Connects parts of a distributed application with "data pipes" and passes data between them

© 2002 David E. Bakken

Middleware in Context



© 2002 David E. Bakken

Middleware: Heterogeneity & Transparency

- Middleware's programming building blocks mask heterogeneity
  - Makes programmer's life much easier!!
- Kinds of heterogeneity masked by middleware
  - Heterogeneity in network technology always masked
  - Heterogeneity in host CPU always masked
  - Heterogeneity in operating system (or family thereof) usually masked
  - Heterogeneity in programming language usually masked
  - Heterogeneity in vendor implementations sometimes masked
- Middleware can provide transparency with respect to distribution:
  - Location transparency
  - Replication transparency
  - Mobility transparency
  - Concurrency transparency
  - Failure transparency
- Masking heterogeneity and providing transparency makes programming distributed systems much easier to do!

© 2002 David E. Bakken

Middleware, MicroQoSCORBA, and Avionics

- Good benefits of productivity and
  - Higher-level building block
  - Less error-prone
  - Analogies: HLL vs. assembler
- Verifiability/validation
  - FAA is picky, picky, picky…..
  - Smaller, much more modular/regularlized middleware such as MicroQoSCORBA likely **much easier to validate** with FAA than a huge, general-purpose CORBA ORB
- Ability to plug in new transports
  - ARINC 664 (nice complement to our ongoing RT work)

© 2002 David E. Bakken

Java

- Java Remote Method Invocation (RMI)
  - Java-centric
  - Lacks cross-language support
- Java VM is ‘large’
  - Java 2 Micro Edition (J2ME)
  - J2ME Connect Device Configuration
    - 2+ MB
  - J2ME Connected Limited Device Configuration
    - 160 MB

© 2002 David E. Bakken

Existing CORBA Specifications

- MinimumCORBA
  - Removes support for dynamic interfaces, etc
  - Reduces the memory footprint of an ORB
- Real-time CORBA
  - Provides tools to better predict time delays
  - Enables hard real-time CORBA applications
- Fault-tolerant CORBA
- CORBAsecurity
- Smart Transducers Interface

© 2002 David E. Bakken

Existing Small Middleware Frameworks

- Support is lacking for
  - Small memory footprint
  - Generality
  - Power awareness
  - Multi-property QoS (esp. non-RT properties)
  - Fine-grained composability
  - Software Engineering & Analysis Tools
- A few “point solutions” with predefined tradeoffs

© 2002 David E. Bakken

Outline

- Embedded systems market background
- Middleware and its benefits
- **MicroQoSCORBA Baseline**
  - **Lifecycle Time Epochs**
    - **Middleware Taxonomy**
  - **Base Architecture**
  - **CASE Tools**
  - **Profiling (work in progress; done Aug. ’02)**
  - **Implementation**
- MicroQoSCORBA Quality of Service (QoS)
  - Fault Tolerance (done July ‘02)
  - RealTime (Work in Progress; done Dec. ‘02)
  - Security (Work in Progress; done Dec ‘02)
  - Possible Boeing-MicroQoSCORBA interactions
- Key Research issues (won’t get to all today….)

© 2002 David E. Bakken

MicroQoSCORBA Bottom Line

- MicroQoSCORBA the only framework that
  - Is a “bottom-up” rethinking from the device level of what should be configurable, and in what ways
  - Is tailorable for a given application to the wide range of
    - Device constraints, ***and***
    - Application-dictated constraints

    with a fine granularity of configuration constraints
  - Will support both “functional” and QoS properties
    - Not just realtime, but security and fault tolerance are also key QoS properties

© 2002 David E. Bakken

Lifecycle Time Epochs

- 1. Initial design
- 2. IDL compilation
- 3. Code compilation/linkage
- 4. System initialization/startup
- 5. Run time execution

© 2002 David E. Bakken

Lifecycle Epoch Table

Lifecycle Epoch	Constraint Bound	Representative Examples
Design	HW Heterogeneity	Symmetric, asymmetric
	HW Choice	X86, TINI, ColdFire
	Communications HW	Ethernet, Serial, Infrared
	Processing Capability	50 Mhz, 1 Ghz, 8bit, 32bit
	System size	small, medium, large (e.g., transducers to jets)
	Power Usage	line, battery, and/or parasitic power
IDL Compilation	Communication Style	Passive, Pro-active, Push, Pull
	Stub/Proxy Generation	Inline vs. library usage
	Message Lengths	Fixed, variable length messages
	Parameter Marshalling	Fixed Formats
Application Compilation	Space/Time Optimizations	Loop unrolling, code migration, function and proxy inlining
	Library Usage	Static vs. dynamic library linkage
System /Application Startup	Device Initialization	
	Network Startup	Bootp, dhcp
	Major QoS adaptation	Select between QoS modules
Run Time	Minor QoS adaptation	Adjust QoS parameters

© 2002 David E. Bakken

1. Initial Design

Embedded Hardware	Roles			SW I/O	IDL Subsetting
	Connection Setup	Data Flow	Interaction Style		
System Components • Homogenous • Asymmetric **HW I/O Support** • Serial, 1-Wire, Parallel, Digital, Ethernet, IrDA, Bluetooth, GSM, GPRS **Resources** • Memory • Power **Processing Capabilities** • 8-bit, 16-bit, …	**Initiates Conn. Setup** **Receive Conn. Requests**	**Bits In** **Bits Out** **Bits In/Out**	**Sync** (Send/Recv) **Async** (One-Way Msgs) **Msg Push** **Msg Pull** **Passive** **Pro-Active**	**Data Representation** • CORBA CDR • MQC CDR • … **Protocols** • TCP/IP • UDP • PPP • 1-wire **Direct** (i.e., IIOP) **Indirect** (i.e., IIOP Gateway)	**Message Types** **Parameter Types** **Data Types** **Exceptions** **Message Payload**

© 2002 David E. Bakken

2. Interface Compilation

- IDL Subsets
  - Verify conformance with design choices
- Application specific code generation
  - Leverage IDL subset meta-data
  - Optimize client stub/server skeleton code
- Representation Optimizations
  - Optimize static string references
  - Object references

© 2002 David E. Bakken

3. Application Compilation/Linkage

- Space/Time optimizations
  - Use existing compiler techniques
- Library usage
  - Static
  - Dynamic
- Future choices (work in progress)
  - MicroQoSCORBA ORB library vs. custom-generated library
  - MicroQoSCORBA Orb library or custom-generated library

© 2002 David E. Bakken

4. System Initialization/Startup

- Device initialization
  - Use reflection to load appropriate modules
  - Initialize QoS subsystems
- Network initialization
  - Locate other key nodes on the network

© 2002 David E. Bakken

5. Run Time Execution

- Fixed functionality
  - Most common case
- Dynamic functionality
  - Increased Resource Usage

© 2002 David E. Bakken

Lifecycle Epoch Table

Lifecycle Epoch	Constraint Bound	Representative Examples
Design	HW Heterogeneity	Symmetric, asymmetric
	HW Choice	X86, TINI, ColdFire
	Communications HW	Ethernet, Serial, Infrared
	Processing Capability	50 Mhz, 1 Ghz, 8bit, 32bit
	System size	small, medium, large (e.g., transducers to jets)
	Power Usage	line, battery, and/or parasitic power
IDL Compilation	Communication Style	Passive, Pro-active, Push, Pull
	Stub/Proxy Generation	Inline vs. library usage
	Message Lengths	Fixed, variable length messages
	Parameter Marshalling	Fixed Formats
Application Compilation	Space/Time Optimizations	Loop unrolling, code migration, function and proxy inlining
	Library Usage	Static vs. dynamic library linkage
System /Application Startup	Device Initialization	
	Network Startup	Bootp, dhcp
	Major QoS adaptation	Select between QoS modules
Run Time	Minor QoS adaptation	Adjust QoS parameters

© 2002 David E. Bakken

MicroQoSCORBA over UDP

Cons of TCP:

- Own model for retransmission of data
- Designed to transmit bulk data; 1bit to indicate "interactive data"; no common equipment checks that bit
- Strictly sequential: no way to access data "as soon as received", or to ignore/discard "obsolete data“ (Ex. You don’t want to retransmit old temperatures from sensors since the temperature may already have changed)
- Kernel resources occupied by TCP stack
  - Typical stack 50-60KB in small devices

© 2002 David E. Bakken

MicroQoSCORBA over UDP (cont.)

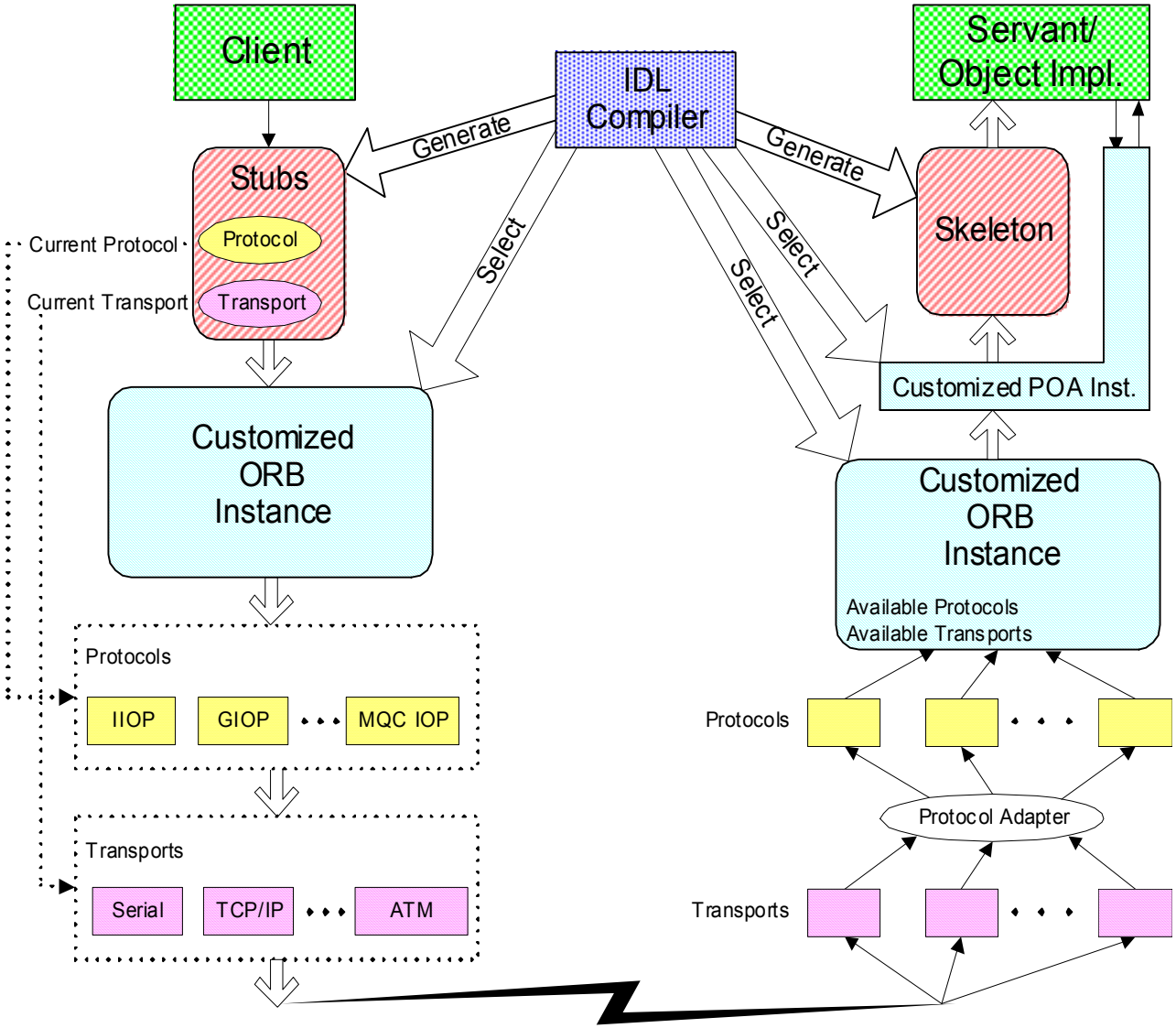
- UDP used to overcome TCP limitations
- Reliability added on top of UDP
- UDP-based protocol needs to be highly aware of data semantics (not just syntax) of what it carries => usage specific

  (Ex. Temperature sensor pinging the temperature every 5 mins. may not need reliability unless certain value turns on fire alarm)

MS thesis on this work completing in July, 2002.

© 2002 David E. Bakken

Baseline Architecture



© 2002 David E. Bakken

Configurability

- Each embedded system is slightly different
- Not everyone will need the kitchen sink
- Let the developer pick and choose
- Configure QoS properties, too
  - Multiple implementations/strengths for one property to choose from (more later in talk)
  - Choose at startup with reflection/introspection

© 2002 David E. Bakken

CASE Tools

- Not all developers are created equal
- Make it easy for the casual programmer
  - Domain expert, but QoS novice
  - Lifecycle support personnel
  - Temporary/contract employees
- Let the tools choose compatible components based upon
  - QoS requirements
  - Resource configuration

© 2002 David E. Bakken

CASE Tools

MicroQoSCORBA Configuration Tool

File Help

Data types | QoS | Comm. & Protocols | Misc

Primary types

boolean, unsigned short, long long, long double, char, wchar, unsigned long long, octet, long, float, short, unsigned long, double

Complex types

struct, union, array, sequence, enum, string, wstring

Exception types

system, user

Open | Save | Check profile | Exit

MicroQoSCORBA Configuration Tool

File Help

Data types | QoS | Comm. & Protocols | Misc

Fault Tolerance
- Retransmit
- Byzantine

Security
- Low
- Medium
- High

Real time
- Soft
- Hard

Network QoS
- Guaranteed BW 0 Mbps
- Low lantecy

Open | Save | Check profile | Exit

MicroQoSCORBA Configuration Tool

File Help

Data types | QoS | Comm. & Protocols | Misc

Client	Server
Transport	Transport
TCP/IP	TCP/IP
UDP	UDP
Serial	Serial
1-Wire	1-Wire
GIOP	GIOP
GIOP	GIOP
GIOP-Lite	GIOP-Lite
MQC-IOP	MQC-IOP

Open | Save | Check profile | Exit

MicroQoSCORBA Configuration Tool

File Help

Data types | QoS | Comm. & Protocols | Misc

Message payload 0 bytes
Max # of interfaces 0
Max # of methods 0

Homogeneous HW
Heterogeneous HW
Force endianess Big endian

Compress Interface/method names

Use inline marshaling/un-marshaling (in a proxy)
Use marshaling/un-marshaling library (in ORB library)

Open | Save | Check profile | Exit

© 2002 David E. Bakken

(Very) Initial Profiling Results (Dec. 01)

```
module sensor {
     interface volts {
          long read_smart_transducer(in long arg1);
     };
};
```

	Class File Size on Linux (bytes)		Executable File Size on TINI (bytes)	
	Client	Server	Client	Server
MicroQoS-ORBA	35,182	43,412	14,598	22,117
ZEN	356,080	351,704	N/A	N/A
JacORB	6,596,196	6,592,640	N/A	N/A

	Average call time (ms)	
	Linux	TINI
MicroQoS-CORBA	0.56	210.00
ZEN	2.03	N/A
JacORB	1.46	N/A

	Memory usage on TINI (bytes)	
	Client	Server
MicroQoS-CORBA	31,840	35,552
ZEN	N/A	N/A
JacORB	N/A	N/A

© 2002 David E. Bakken

Future Profiling Directions

- Memory footprint
  - Total vs. RAM vs. ROM
  - Growth rate w.r.t. parameters, methods, …
- Timing & CPU utilization
  - End-to-End latency & jitter
  - Hot spot analysis
    - Network I/O, Parameter marshalling, Buffer management
- Power consumption
- Non-Java language (C++) support in some cases/ configurations

© 2002 David E. Bakken

Outline

- Embedded systems market background
- Middleware and its benefits
- MicroQoSCORBA Baseline
  - Lifecycle Time Epochs
    - Middleware Taxonomy
  - Base Architecture
  - CASE Tools
  - Profiling (work in progress; done Aug. ’02)
  - Implementation
- **MicroQoSCORBA Quality of Service (QoS)**
  - **Fault Tolerance (done July ‘02)**
  - **RealTime (Work in Progress; done Dec. ‘02)**
  - **Security (Work in Progress; done Dec ‘02)**
  - **Possible Boeing-MicroQoSCORBA interactions**
- Key Research issues (won’t get to all today….)

© 2002 David E. Bakken

Quality of Service (work in progress)

- Quality of Service (QoS): the “non-functional” parts of the program
  - “functional” part is the “what”
  - “non-functional” part is the “how”
- Security
  - Multiple strengths/Algorithms
- Fault Tolerance
  - Quantity and types of faults tolerated
- Real-time Behavior
  - Scheduling Algorithms, Network performance
- Resource Issues
  - Memory footprint
  - Power awareness
- Which combos of QoS modules are legal???

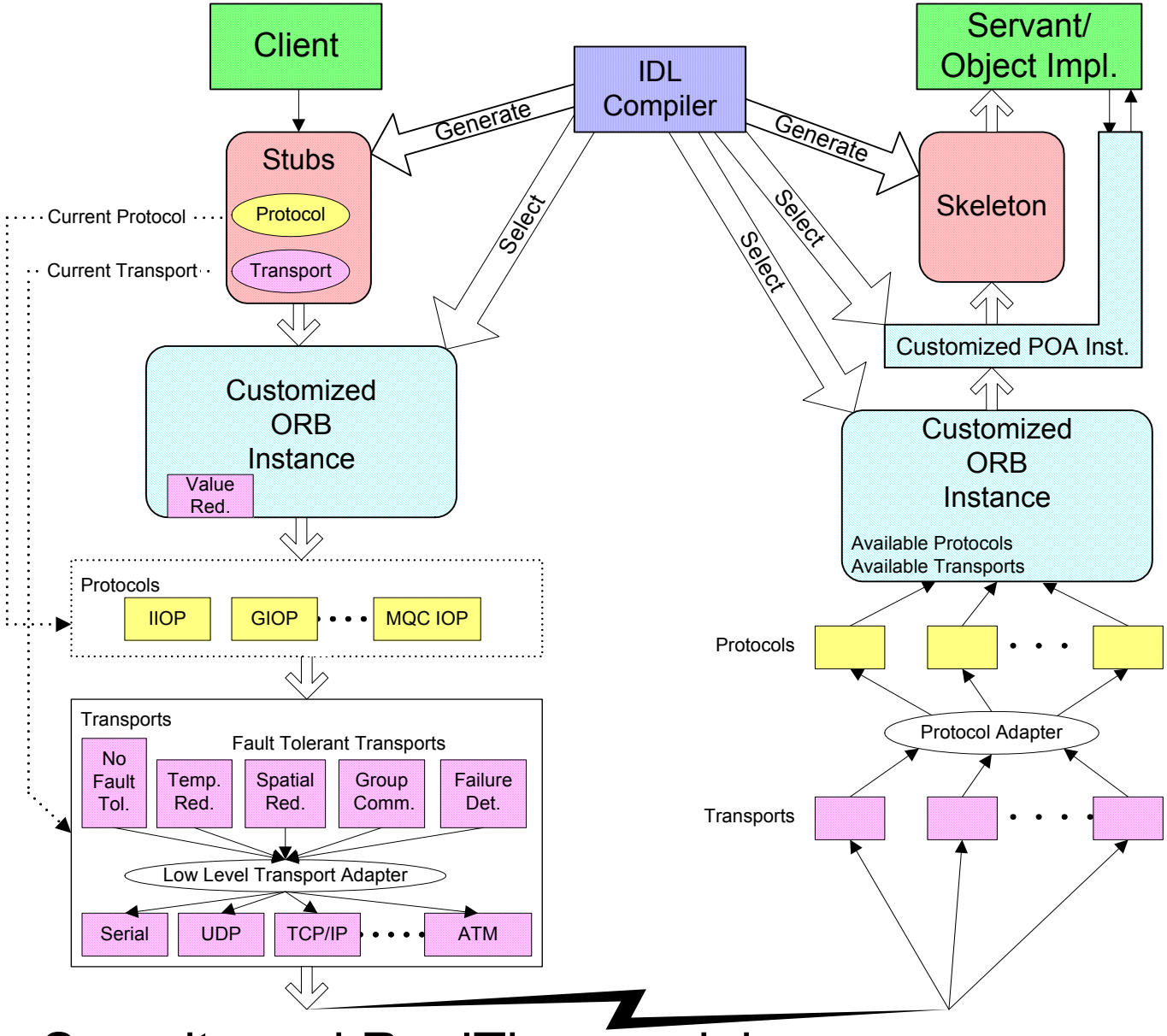
© 2002 David E. Bakken

Fault Tolerance Taxonomy

Redundancy	Reliability	Ordering
•Temporal –Multiple transmits •Spatial –Mutiple Channels –Multiple servers •Value –Checksums, CRC	•Group Communication –Best Effort –Reliable –Uniform –Atomic •Failure Detection	•Sender FIFO •Causal –Logical Timestamping •Total –Sequencer / Token based

© 2002 David E. Bakken

MicroQoSCORBA Initial QoS Architecture



- Note: Security and RealTime work in progress….

© 2002 David E. Bakken

MicroQoSCORBA, RealTime, and Boeing

- Real-Time research
  - Profiling of components and application for timing, not memory (as current profiling is)
  - 1-2 CPU scheduling mechanisms TBD
  - MS thesis work, finishing in Dec. 2002 by Dr. Wesley Lawrence
- Possible Boeing and Real-Time collaboratory project
  - Add ARINC 644 (or subset(s) thereof?) to MicroQoSCORBA
    - Nice Real-Time network mechanisms to augment above MS work!
  - Target a few extra specific processors

© 2002 David E. Bakken

Conclusions

- MicroQoSCORBA the only framework that
  - Is a “bottom-up” rethinking from the device level of what should be configurable, and in what ways
  - Is tailorable for a given application to the wide range of
    - Device constraints, ***and***
    - Application-dictated constraints

    with a fine granularity of configuration constraints
  - Will support both “functional” and QoS properties
    - Not just realtime, but security & fault tol. also key QoS properties
- MicroQoSCORBA now interacts with TAO ORB, others easy (via IIOP)
  - Access to naming service, trader service, fault tolerance service (non-transparent), etc…..
  - Access to bigger servers and the world at large!
- Questions? Comments? Potential Collaborations?

© 2002 David E. Bakken

BACKGROUND SLIDES [Outline]

- Embedded systems market background
- Related middleware research
- MicroQoSCORBA
  - Lifecycle Time Epochs
    - Middleware Taxonomy
  - Quality of Service
  - Architecture
  - CASE Tools
  - Implementation
- **Key Research issues**

© 2002 David E. Bakken

Outline

- Embedded systems market background
- Middleware and its benefits
- MicroQoSCORBA Baseline
  - Lifecycle Time Epochs
    - Middleware Taxonomy
  - Base Architecture
  - CASE Tools
  - Profiling (work in progress; done Aug. ’02)
  - Implementation
- MicroQoSCORBA Quality of Service (QoS)
  - Fault Tolerance (done July ‘02)
  - RealTime (Work in Progress; done Dec. ‘02)
  - Security (Work in Progress; done Dec ‘02)
  - Possible Boeing-MicroQoSCORBA interactions
- **Key Research issues (won’t get to all today….)**

© 2002 David E. Bakken

Key Questions

- Q1: Baseline Constraints:
  - What constraints must be imposed on middleware to achieve a small footprint (ignoring QoS for now)?
- Q2: QoS Properties:
  - What strength or "flavor" of QoS properties (security, fault tolerance, real-time) can reasonably be supported in resource-starved embedded systems?
- Q3: Network-Wide Composition:
  - What issues, optimizations, and tradeoffs must be made when a network of QoS-enabled devices integrated? How can we support global properties, metrics, and goals involving the entire distributed embedded system?
- Q4: Software Engineering Issues:
  - How can we use QoS aspects to make the middleware more manageable and quantifiable? How can we use patterns and tools to help the developer to be more productive?

© 2002 David E. Bakken

Key Questions in Context

Embedded Systems
Application Developer

Q4: ***Software Engineering***

Aspects
Patterns
Tools

Application Logic

Application Logic

Application Logic

Q2: ***QoS***
Security, Realtime,
Fault Tolerance

Q1: ***Baseline Functionality***

Power | ··· | Memory

Node 1

2

3

Q3: ***Local-Global Interactions***

© 2002 David E. Bakken

Question 1: Baseline Functionality

- What constraints must be imposed on middleware to achieve a small footprint (ignoring QoS for now)?
  - 1a. Architecture
  - 1b. Costs
  - 1c. Models
  - 1d. Composition
  - 1e. Bottom line

© 2002 David E. Bakken

1a: Architecture

- What middleware components or subsystems should be constrained?
- What should their architecture be?
  - Covered previously in this presentation

© 2002 David E. Bakken

1b: Costs

- What memory footprint and power costs are associated with each subsystem or facet?
  - What HW hooks are available for gathering profiling information?
  - Can SW profiling hooks be added?
  - Can rough, empirical rules-of-thumb be derived for cost estimation?
- How much can be gained by constraining a subsystem's flexibility or removing it?

© 2002 David E. Bakken

1c: Models

- What models and heuristics for predicting memory and power usage can be developed, to map from larger environments with adequate profiling and other “hooks” onto very small embedded environments where this is not possible?
- How can spreadsheet-like online tools use these models to allow developers to project the costs of various possible tradeoffs and constraints without having to code them (like with 1b)?

© 2002 David E. Bakken

1d: Composition

- How do combinations of these constraints compose or interact?
- What combinations are legal or useful?
- How do we best allow a developer to constrain them?
  - Where in the SW architecture will these constraints be implemented?
  - Do the HW choices impose mutually exclusive constraints?
  - What constraints will provide the greatest benefit?
  - Must some constraints be implemented in order to support QoS?
  - Should the CASE tool
    - allow the developer to have access to all constraint choices
    - provide a standard set of design constraint patterns

© 2002 David E. Bakken

1e: Bottom Line

- How small and power stingy can we get, and with what constraints?
  - How can we instrument the HW/SW?
    - Are hooks provided for memory usage?
    - Can we gather power consumption information?
    - Can a SW code analysis provide the required information?
  - How does the choice of HW cpu/platform affect small size/power?
    - Battery power, parasitically powered devices, …
  - What size/power trade-offs exist?
    - Batching messages saves power but increases code size
    - How much ‘hand tweaking’ of the code is justifiable at the cost of maintainability?

© 2002 David E. Bakken

Question 2: QoS Properties

- What strength or “flavor” of QoS properties (security, fault tolerance, realtime) can reasonably be supported in resource-starved embedded systems?
  - 2a. Mechanisms
  - 2b. QoS Subsets
  - 2c. Costs
  - 2d. Models
  - 2e. Qos Composition
  - 2f. Composition with Baseline

© 2002 David E. Bakken

2a: QoS Mechanisms

- What mechanisms are used to support each QoS property?
  - What HW design choices ensure QoS or detract from it?
    - eg, dedicated transmission lines ensure message confidentiality whereas wireless links are open to eavesdroppers
    - What are the costs of SW vs. HW QoS implementations?
  - Security
    - Physical protection, Data encryption, Access control, Digital signatures, A/Symmetric encryption, Physical tokens, Shared secrets, Challenge/Response protocols, Identity attributes, Privileges
  - Fault Tolerance
    - Simple retransmits, Passive vs. Active replication, Redundant HW
  - Real-Time
    - Static vs. Dynamic scheduling, Determinism, Jitter control

© 2002 David E. Bakken

2b: QoS Subsets

- What subsets of these mechanisms exist for each property that is reasonable in resource-starved environments?
  - Is HW support required for some mechanisms?
  - How much hardware support is required for each mechanism?
  - How many interchangeable implementations should be developed for a given QoS sub-property?
  - Are some subsets effectively mutually exclusive?
  - How many subsets should be exposed to the developer?
  - Can we derive reasonable Subset Design Patterns?

© 2002 David E. Bakken

2c: QoS Costs

- What are the costs of these mechanisms?
  - Can we estimate (pre-implementation) costs of a QoS mechanism?
  - What HW costs exist?
  - What SW costs exist?
  - Does HW support exist for profiling the QoS Costs?
  - Do we need to implement SW hooks for gathering profiling/cost information?
  - When do the increased costs become prohibitive?
  - Is there a point where the application should be transitioned to a full-featured ORB?

© 2002 David E. Bakken

2d. Models

- What models and heuristics for predicting QoS costs can be developed, to map from larger environments with adequate profiling and other “hooks” onto very small embedded environments where this is not possible?
- How can spreadsheet-like online tools use these models to allow developers to project the costs of various possible tradeoffs and constraints without having to code them?

© 2002 David E. Bakken

2e: QoS Composition

- What combinations of these mechanisms are useful for resource-starved environments?
  - What are some common ‘Use Cases’ that require multi-property QoS?
  - Can we develop some useful design patterns for these ‘Use Cases’?
  - At what level (or multiple levels?) should these multi-property QoS combinations be developed/described?
  - Are the resource constraints for combinations additive?

© 2002 David E. Bakken

2f: QoS Interactions

- What interactions do the QoS subsystems have with the functional subsystems and constraints?
  - Which functional and QoS subsystems are co-located?
  - Which functional and QoS subsystems (or the implementation of their mechanisms) are orthogonal?
  - Do some mechanisms compose better with the functional aspects than others?
  - Are some subsystem interactions mutually exclusive?

© 2002 David E. Bakken

Q. 3: Network-Wide Composition

- What issues, optimizations, and tradeoffs must be made when a network of QoS-enabled devices integrated?
- How can we support global properties, metrics, and goals involving the entire distributed embedded system?
  - 3a. Local-Global Interactions

© 2002 David E. Bakken

3a: Local-Global Interactions

- What basic issues, optimizations, and tradeoffs are associated with networks of QoS enabled embedded systems?
  - What new metrics need to be developed?
  - Can existing metrics be QoS-parameterized?
- How does power awareness impact the local-global interactions?

© 2002 David E. Bakken

Q. 4: Software Engineering Issues

- How can we use QoS aspects to make the middleware more manageable and quantifiable?
- How can we use patterns and tools to help the developer to be more productive?
  - 4a. Aspects
  - 4b. Patterns
  - 4c. Tools

© 2002 David E. Bakken

4a: Aspects

- How can the QoS properties in Question 2 be organized into aspects that are orthogonal?
- How can this be woven cleanly in the architecture?
  - Aspect-oriented IDL compiler generated code
  - How will aspect weaving affect the cost models produced in Questions 1 & 2

© 2002 David E. Bakken

4b: Patterns

- What canonical patterns involving combinations of functional constraints and QoS requirements are common in various embedded system domains?
  - What are the common use cases?
- How can these be organized to help middleware novices avoid choosing from 2N or similar number of combinations?

© 2002 David E. Bakken

4c: Tools

- An initial set of tools has been developed.
- How can the tools be extended in order to encompass the cost projections and models developed in Questions 1 & 2?
- How can the tools present design patterns to the software developer?

© 2002 David E. Bakken

Optional/Related Questions

- What middleware services (naming, etc.) and variants thereof are useful for embedded systems, and what are their costs?

© 2002 David E. Bakken

Conclussions

- MicroQoSCORBA the only framework that
  - Is a “bottom-up” rethinking from the device level of what should be configurable, and in what ways
  - Is tailorable for a given application to the wide range of
    - Device constraints, ***and***
    - Application-dictated constraints
    - with a fine granularity of configuration constraints
  - Will support both “functional” and QoS properties
    - Not just realtime, but security and fault tolerance are also key QoS properties
  - Will support the Developer with SW Eng. Tools

© 2002 David E. Bakken