CS4220 Embedded Systems
CS6235 Real-Time Systems

4B: QoS and IDL

Instructor: Calton Pu

calton.pu@cc

TA: Younggyun Koh (young@cc)

Hard RT vs. Soft RT

- Traditional view of real-time
  - Complete predictability (Hard RT)
- New time-critical applications
  - Data-intensive applications (e.g., multimedia)
  - Not a dichotomy, but a spectrum/continuum
- The entire system working together
  - All resources and trade-offs among them

What Is QoS?

- Traditional QoS
  - Example: networking QoS
  - Dichotomy: a task either succeeds or fails
- Quality of Service (informally)
  - Task may achieve partial success at deadline
  - If it's a periodic task, let's start the new task
  - System performance (QoS) as an integration of partial/total successes and failures

Broad View of QoS

- Involves all resources
  - CPU, memory, disk I/O, network, power …
- In general, you won't get guarantees
  - mobility, wireless networks …
  - increasingly heterogeneous hardware and software
  - increasingly high quality stored data
- QoS about specifying adaptation policies
  - A much richer space for QoS semantics

Why Is QoS Important?

- Multimedia presentation requirements exceed available resource capacity
  - presentation requirements are increasing
  - range of resource capabilities is widening
- In general, resource saturation
  - Transient, sudden demand
  - Natural evolution of steady state load
  - Denial of service attacks

Cactus Highlights

- Middleware support for QoS
  - QoS is an end-to-end property
  - OS typically does not support QoS directly (Discussion on the difficulties of OS support)
  - Applications also have difficulties
  - Portability is a major goal
  - Actually, sitting on top of normal middleware

CQoS Architecture

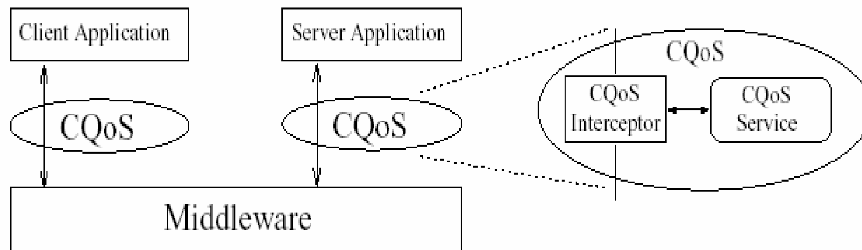


Figure 1: High-level view of CQoS architecture.

QoS Properties

- Concrete QoS properties
  - Network: response time and bandwidth
  - CPU: total amount in each period
- Abstract QoS properties
  - Performance (timeliness, fulfillment)
  - Fault tolerance (automated recovery)
  - Security, etc

CQoS Components

- CQoS Interceptor
  - Specific to underlying middleware platform and to applications using CQoS
  - Interface to and wrapper of CQoS services
- CQoS Service
  - Provides support for abstract QoS properties

CQoS Interceptors

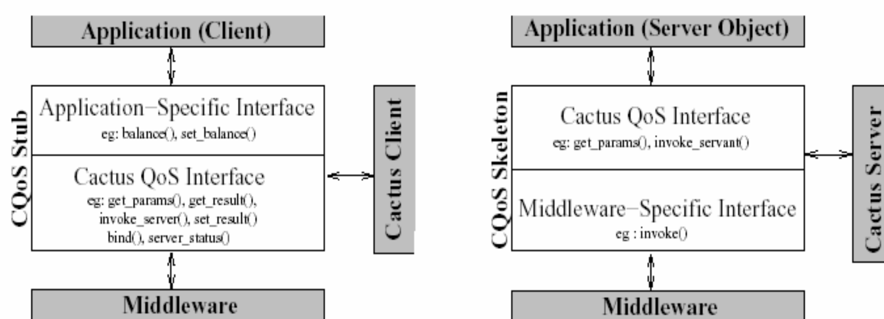


Figure 2: Structure of CQoS interceptors.

Interceptor Middleware

- What is middleware?
  - Code inserted between layers of software
- Downward calls
  - Uses underlying middleware facilities
  - Adds Cactus QoS interface
- Upward calls
  - Application-specific interface
- IDL stubs (thin, compiler-generated code)

Service Component

- Micro-protocol
  - Simple module that implements one function
  - Carefully defined interface for composition
- Composite protocols
  - Combinations of micro-protocols and their functionality
  - Works when combination works
- In contrast to Aspect-Oriented Progr. (AOP)

Cactus Customization

- Static customized composition
  - Constructor of composite protocol specifies which micro-protocols are needed
- Dynamic customized composition
  - Java dynamic loading
  - Server and client my download appropriate micro-protocols according to configuration

Implementing QoS

- Base micro-protocols (communications)
- Fault-tolerance QoS properties
  - Active replication: eager symmetric execution
  - Passive replication: primary-secondary
  - First answer, first success, majority voting
- Security QoS properties
  - Message confidentiality, integrity, access control

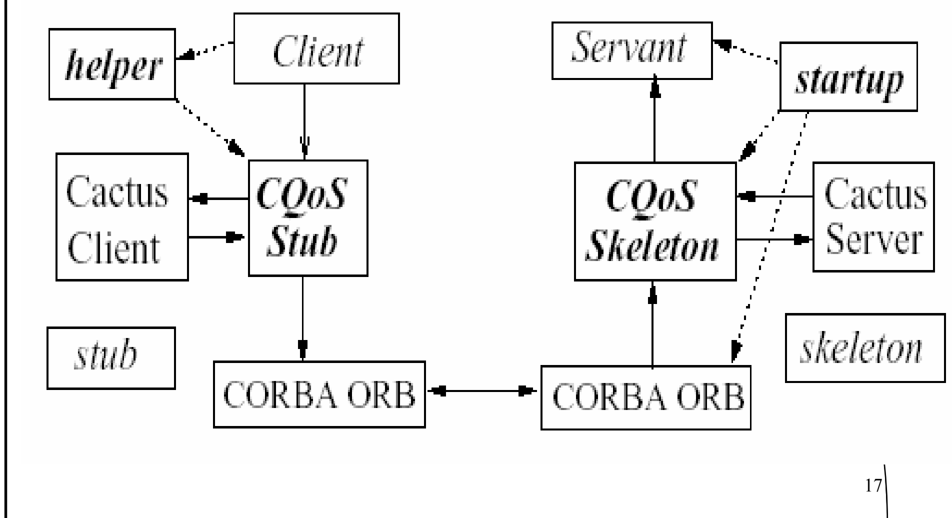
Performance QoS

- Timeliness properties
  - DiffServ: more timely service for high priority requests
  - PrioritySched: manages thread priorities
  - QueuedSched: manages wait queue
  - TimedSched: refinement of queue mgmt
- Some complications with priority inversion
  - Need careful combination of micro-protocols

Composite Protocols

- Easy combinations
  - When the dimensions are orthogonal
- Interferences and trade-offs in composition
  - Same resource: response time and bandwidth trade-offs in networks
  - Difference resources: CPU and network bandwidth utilization in adaptive compression
  - Security (heavy encryption) and throughput (light encryption)

Cactus Components



Summary of Cactus

- Good ideas
  - Micro-protocols (from *x*-kernel project)
  - Customization & composition (several projects)
  - IDL and middleware (several projects)
  - Interception and services (ongoing trend)
- Some limitations
  - Only Orthogonal QoS properties (AOP work)
  - "Demo only" software

Middleware Foundations

- The simpler good ol' days
  - Applications run on top of OS
  - Maybe there is a DBMS
- Distributed HW/SW components
  - Difficult to connect the components
  - Difficult to allocate resources
  - Need more "glue" code and resource mgmt

RPC Stub Generator

- Remote Procedure Call
  - Major advance (early 80's) in distributed prog.
  - Encapsulate remote function into RPC
  - Automates interfacing code generation
- RPC Stub Generator
  - Translates a procedural interface into messages
  - Marshalling and unmarshalling of parameters, with type checking

IDL Compiler

- Interface Definition Language
  - A domain specific language
  - Declarative specification of method invocations (usually procedural interfaces of RPCs)
  - Generalization of RPC stub generator
- IDL compiler creates invocation code
  - Can translate into RPC/RMI
  - And other formats, and code for a variety of underlying "abstract machines"

Simple Examples

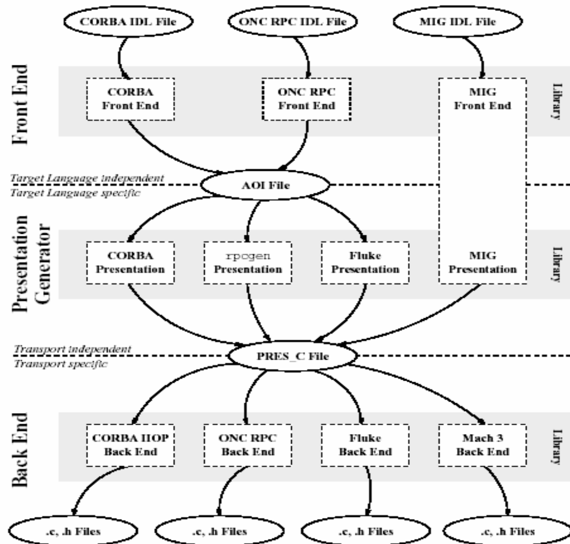
cation. For example, the following CORBA [18] IDL program declares a simple interface to an electronic mail service:

```
interface Mail {
    void send(in string msg);
};
```

A largely equivalent mail system interface would be defined in the ONC RPC[1] [23] IDL by this program:

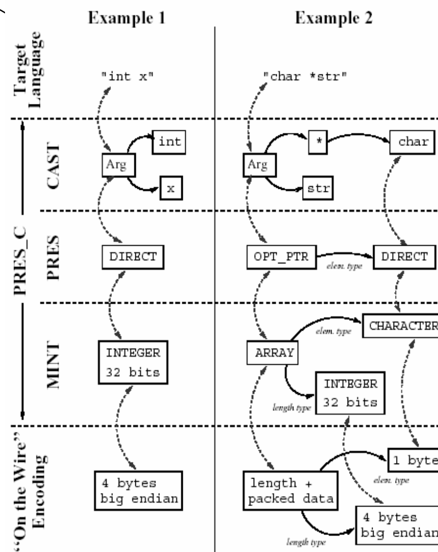
```
program Mail {
    version MailVers {
        void send(string) = 1;
    } = 1;
} = 0x20000001;
```

Flick Architecture

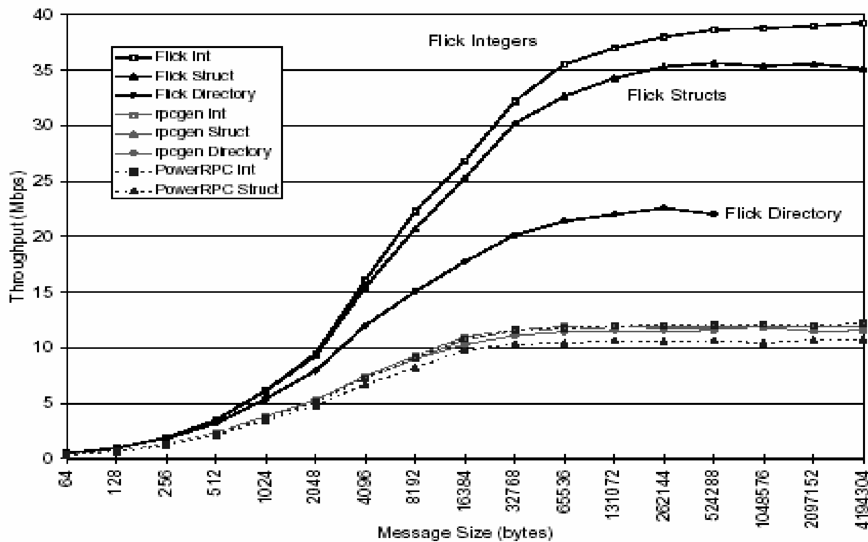


Translation Process

- Multi-stage translation
  - CAST: C abstract syntax tree
  - PRES: message presentation
  - MINT: message interface
- Domain-specific compiler optimizations at back-end



Better Performance



Summary of Flick

- Good idea: compiler techniques for IDL
  - Good intermediate representations supporting multiple input formats and output formats
  - Good optimization methods supporting the generation of very efficient code
  - Good performance evaluation and graphs
- Some limitations
  - Compiler techniques vs. Axpects