Research Directions for Middleware

Douglas C. Schmidt
schmidt@cs.wustl.edu

Washington University, St. Louis
www.cs.wustl.edu/~schmidt/

January 23rd, 1999
Overview of Middleware

- Observations
  - Historically, apps built directly atop OS
  - Today, more and more apps built atop middleware
  - Middleware has several layers

- General Research challenges
  - Performance optimizations
  - Quality of Service (QoS)
  - Software architecture & patterns
Scope of Performance Optimization Challenges

CLIENT

OBJ REF

operation()

in args

out args + return value

OBJECT (SERVANT)

IDL STUBS

ORB INTERFACE

IDL SKELETON

OBJECT ADAPTER

ORB CORE

GIOP

OS KERNEL

OS I/O SUBSYSTEM

NETWORK INTERFACES

NETWORK

PRESENTATION LAYER

DATA COPYING & MEMORY ALLOCATION

SCHEDULING, DEMUXING, & DISPATCHING

CONCURRENCY MODELS

TRANSPORT PROTOCOLS

I/O SUBSYSTEM

NETWORK ADAPTER

CONNECTION MANAGEMENT
**Scope of QoS Challenges**

- **Key Challenges**
  - Specifying QoS requirements
  - Determining operation schedules
  - Alleviating priority inversion and non-determinism
  - Reducing latency/jitter for demultiplexing
  - Reducing presentation layer overhead
  - Maintaining small footprint

---

1) **CLIENT MARSHALING**
2) **CLIENT PROTOCOL QUEUEING**
3) **NETWORK DELAY**
4) **SERVER PROTOCOL QUEUEING**
5) **THREAD DISPATCHING**
6) **REQUEST DISPATCHING**
7) **SERVER DEMARSHALING**
8) **METHOD EXECUTION**
Example: Providing QoS to Remote 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

```c
struct RT_Info {
    Time worstcase_exec_time_;  
    Period period_; 
    Criticality criticality_;    
    Importance importance_;    
};
```
Scope of Software Architecture Challenges

- **Solution Approach**
  - Integrate RT dispatcher into ORB endsystem
  - Support multiple request scheduling strategies
    - *e.g.*, RMS, EDF, and MUF
  - Requests ordered *across* thread priorities by OS dispatcher
  - Requests ordered *within* priorities based on *data dependencies* and *importance*
Applying Patterns and Frameworks to Middleware

• Benefits
  – Facilitate design and code reuse
  – Preserve crucial design information
  – Guide design and implementation choices
  – Document and alleviate common traps and pitfalls

www.cs.wustl.edu/~schmidt/ORB-patterns.ps.gz
Summary of Communication Middleware Research

- **Current generation: real-time middleware**
  - Real-time static scheduling services
  - Minimize ORB priority inversion and non-determinism
  - Reduced end-to-end latency via demuxing optimizations
  - Applied optimizations to IIOP protocol engine
- **Future work**
  - Dynamic and hybrid scheduling of distributed remote operations
  - Distributed QoS and integration with high-speed networks
  - Optimizing IDL compilers
Concluding Remarks

- Researchers and developers of distributed, real-time applications confront common challenges
  - *e.g.*, service initialization and distribution, error handling, flow control, scheduling, event demultiplexing, concurrency control, persistence, fault tolerance
- Successful solutions apply *design patterns, frameworks*, and *components* to resolve these challenges
- Middleware is an effective way to achieve reuse of distributed software components
- Requirements for next-generation communication middleware provide a fertile source of research topics