Adaptive Middleware

Self-Healing Systems
Guest Lecture

Prof. Priya Narasimhan
Assistant Professor of ECE and ISRI
Carnegie Mellon University

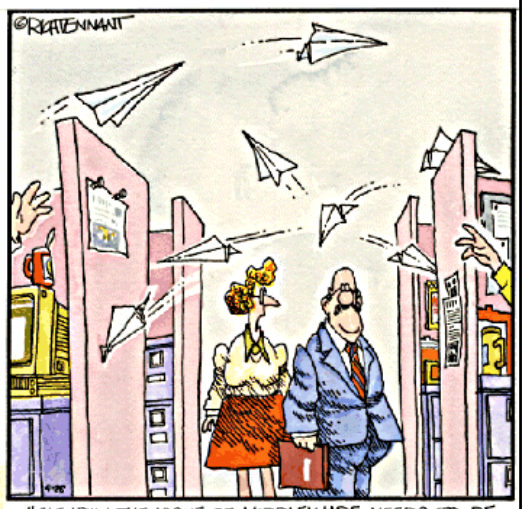
Recommended readings and these lecture slides are available on CMU's BlackBoard

Electrical & Computer ENGINEERING

ISRI

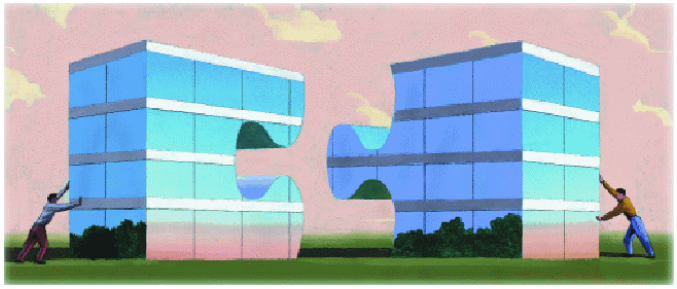
Today's Lecture

- Overview of adaptive middleware (the three papers we selected)
- Behavioral/structural adaptation through Interceptors
  - Running theme across all middleware (and our three papers)
- Open issues and discussion
  - Needs for adaptive middleware
  - Mechanisms
  - Research issues

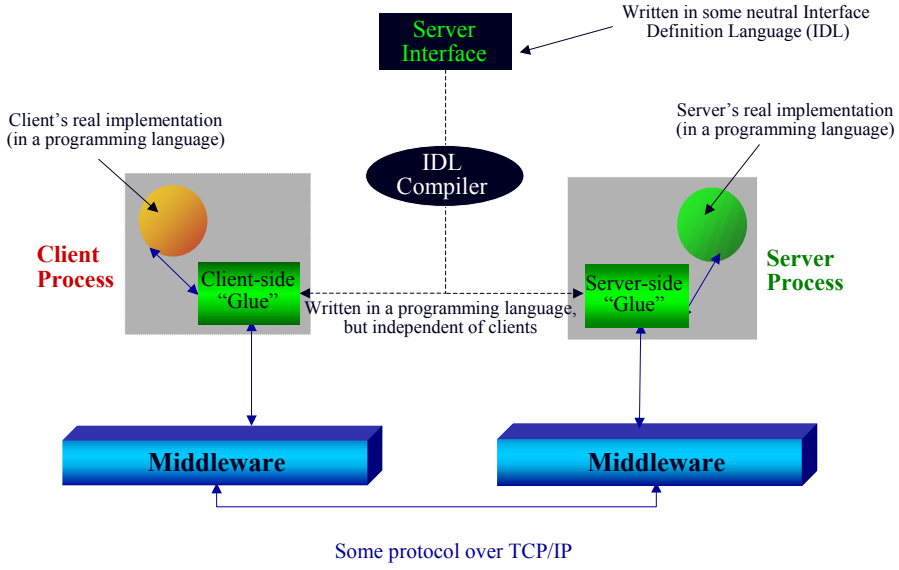


….. And Middleware was Born

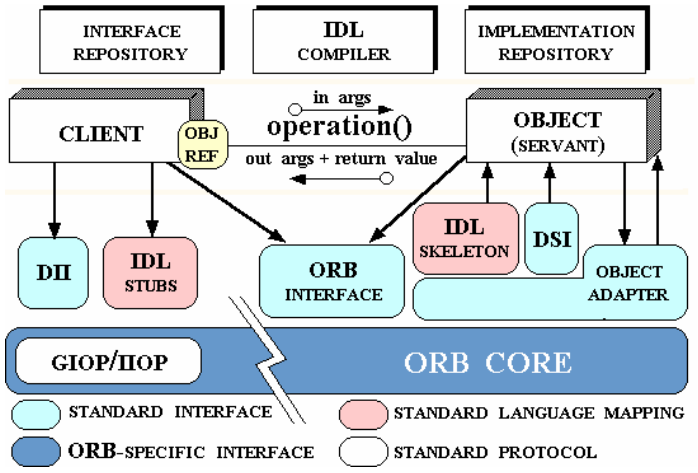
- Why is it called “middleware”?
  - It’s right in the middle of a client and a server
  - Hides operating system and low-level details from the application developer
  - Also called “distributed object computing”
    - CORBA, EJB, DCOM
- Why do we need/have middleware?
  - It makes it easier to write distributed applications
  - Takes care of all the networking code and the messaging
  - Leaves you free to focus on writing the application



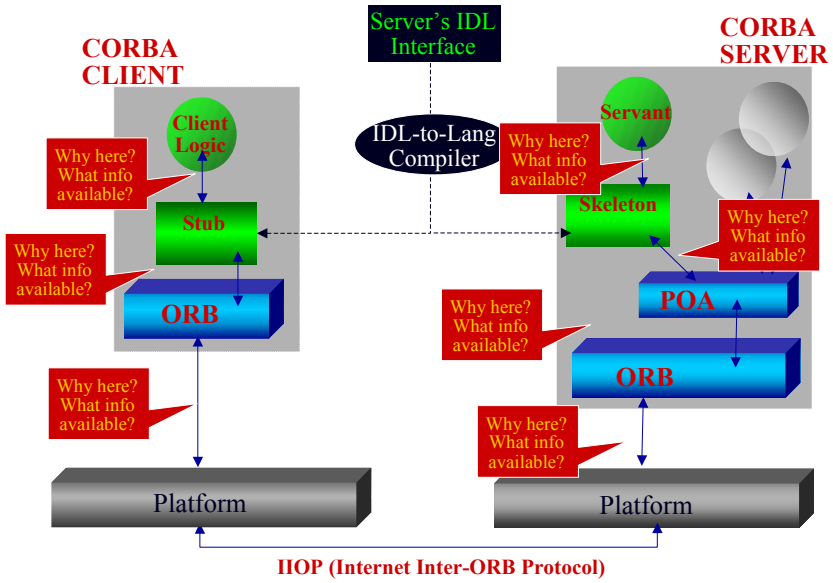
Middleware – The Big Picture



CORBA - Putting All The Pieces Together



Points of Instrumentation/Monitoring/Adaptation

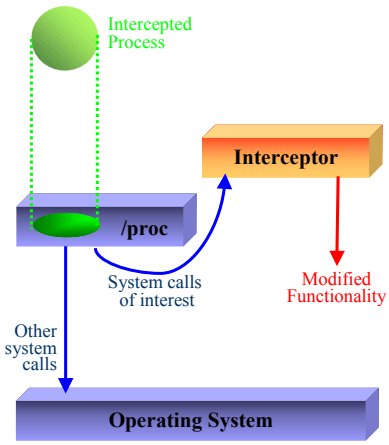


Interceptors – Monitoring/Adaptation Mechanism

- Interceptor - Application-level/Middleware-level extension
- Running theme of all middleware adaptation schemes
  - Intercepts calls/messages to add new components or to modify behavior
  - Mechanism for instrumentation, monitoring, behavioral/structural adaptation
- Can be exploited to enhance an application with monitoring, security, protocol adaptation, fault tolerance, etc.
- External interceptors (outside the middleware and application)
  - Exploits operating system hooks for transparent interception
  - Process control mechanisms such as */proc* in Unix
  - Library interpositioning in Unix and WindowsNT
- Compiled-in interceptors (within the middleware and application)
  - Portable Interceptors available within CORBA and Java middleware
  - Application developer installs/configures code for adaptation

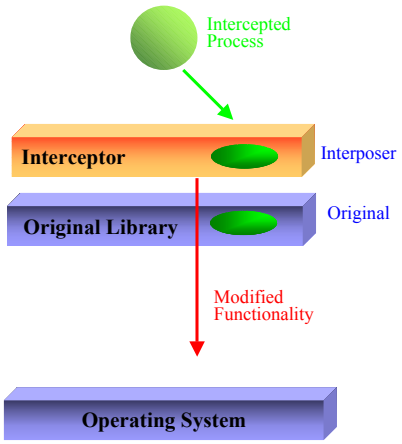
External Interceptors: The /proc Interface

- System call interception
- Interceptor is an external "catcher" process
- Can attach to a process (with process identifier *pid*) by monitoring */proc/pid*
- Specified system calls can be captured at entry and exit
- Can extract and modify
  - Return values of calls
  - Return values of arguments
- Commercial monitors use this technique
  - strace on Linux
  - truss on Solaris



External Interceptors: Library Interpositioning

- Library routine interception
- Interceptor is a library collocated with the intercepted process
- Can override the default definitions of routines in the intercepted process' libraries
- Can extract and modify
  - Return values of routines
  - Arguments of routines
  - The semantics of routines
- Applied in commercial operating systems
  - NTWrappers on Windows
  - LD_PRELOAD on Linux and Solaris



Compiled-In Interceptors

- CORBA and Java contain a specification for Portable Interceptors
  - Can add data to requests/responses
  - Can observe sender identity, request parameters, return value
  - Can be used for fault injection, resource management, load balancing
- Allows you to modify a request on the client-side before transmission
- Allows you to modify a response on the server-side before transmission
- For tracing memory, performance and other resource usage, you have to rely on operating system-provided tools

Why These Three Papers?

- *The Case for Reflective Middleware*
  - High-level overview of reflection as a technique for adaptation of various kinds
  - Three distinct points of adaptation
    - Architecture, interception, resources
- *Comparing and Contrasting Adaptive Middleware Support in Wide-Area and Embedded Distributed Object Applications*
  - Drills down to a specific set of QoS adaptation mechanisms
    - Contract definition and enforcement; gateways (compiled-in interceptors)
- *Strong Replica Consistency for Fault-Tolerant CORBA Applications*
  - Specific kinds of adaptations for one kind of QoS (fault-tolerance)
    - Run-time external interceptors for transparent adaptation
- Can we do fault-tolerant adaptation with any of these three approaches? Yes!
  - FRIENDS (reflection-based), AQuA (QuO-based) and Eternal (interceptor-based)

The Case for Reflective Middleware

- Self-representation of middleware
  - Explicit representation of internal middleware structure maintained by the middleware itself (self-aware middleware)
- Middleware organized as a group of collaborating components
  - Used to build low-footprint ORBs
  - Used for dynamic customization of component behavior
- Two implementations: DynamicTAO and Open ORB
- Architecture/Interface reflection
  - Structural adaptation using a component graph to represent the interconnections between components, along with a set of architectural constraints
- Interceptor-based reflection
  - Behavioral adaptation by pre- and post- processing of interactions
- Resource-based reflection
  - Behavioral/structural adaptation by access to platform's resources

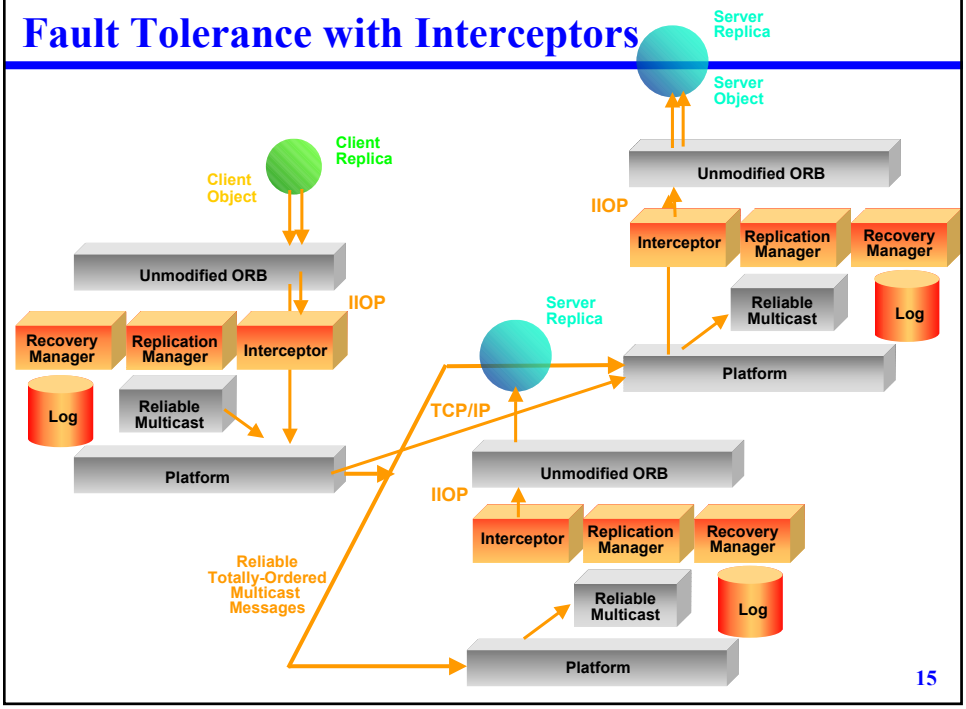
Quality Contracts – QuO

- Allows middleware developers to specify
  - QoS requirements through contracts
  - System elements to be monitored
  - Behavior for run-time adaptation
- Quality Description Language (QDL)
  - High-level language for defining QoS aspects of applications and the adaptive behavior of objects
- QuO runtime kernel
  - Evaluation of contracts and monitoring of objects
- Code generators
  - Weaving of QDL descriptions with the QuO kernel code to produce the application
- Resource-oriented focus: bandwidth, CPU, data size, image quality, etc.
  - Flexibly adaptable to different environments (tightly-constrained real-time applications, as well as heterogeneous enterprise applications)

The Eternal System

- Works with any open standard middleware
  - J2EE/Java and CORBA
  - Replicates application objects for fault tolerance
  - Maintains strong replica consistency through mechanisms for replication, logging and recovery
- Mechanisms implemented underneath the middleware for reasons of efficiency
- Exploits external interceptors to provide fault tolerance transparently
- Extensions of this work for adaptive middleware
  (not contained in the paper – ask me if you are interested)
  - Transparent security for middleware applications – enables middleware applications to adapt to malicious faults/intrusions in the system, without losing service/dependability
  - Live software upgrades – enables middleware applications to be upgraded while they are running, without losing service/reliability

Fault Tolerance with Interceptors



Critical Thinking

- What are the hooks for adaptive middleware currently?
  - Are they sufficient? If so, for what?
- What seems to be missing for adaptive middleware?
- What kinds of events/phenomena can middleware adapt to currently?
- What kinds of events/phenomena should future middleware adapt to?
- What are the goals for adaptive middleware today?
- Are the adaptive middleware techniques applicable to other self-healing environments and systems?
- How can adaptability contracts/requirements be communicated to middleware?
- At what level(s) of the middleware should adaptability be introduced?
  - What are the trade-offs in introducing the adaptive mechanisms at different levels?
- What are the metrics for a "good" adaptive middleware system?
- What are the resources of interest for an adaptive middleware system?
- What about conflicting adaptations; how do we express them and deal with them, particularly at lower levels of the middleware?

- What are the current open research issues in adaptive middleware?

Conclusion

- Adaptive middleware
  - Mechanisms like interceptors exist
  - Already incorporated into standards and current implementations
- Three different papers discussed
  - Reflection
  - QoS contracts
  - Interception
- Several open research issues
- My current research
  - Making middleware adapt to real-time, security and fault-tolerance needs simultaneously in a distributed environment

Adaptation

Not Just for Software Engineers!