Meta-Programming Techniques to Configure Open Standard RT CORBA Middleware Declaratively

Joseph K. Cross, Ph.D.

Lockheed Martin Tactical Systems

Problem: Applications Depend on Middleware Products

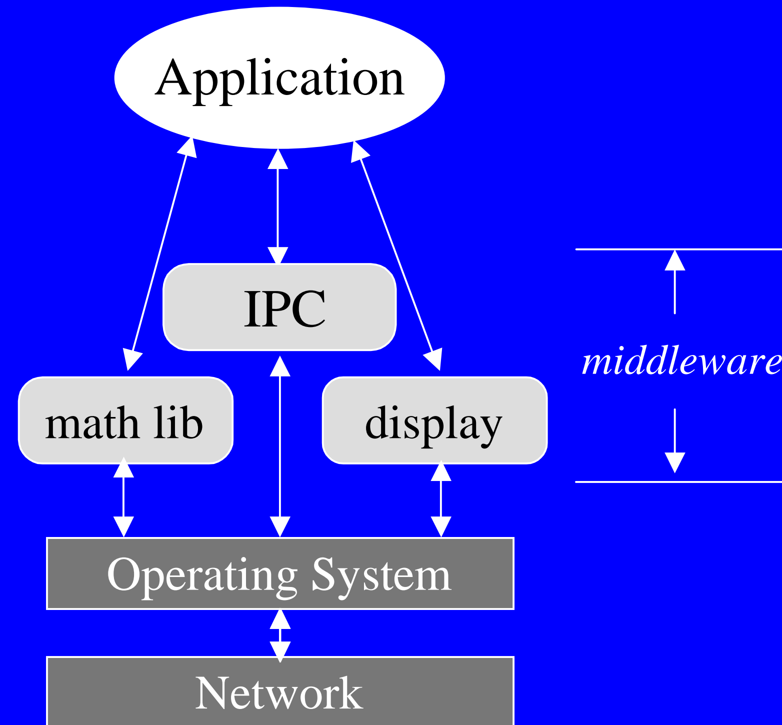
Dependency:
If middleware product changes, then application source code must change

Avoided by use of standard interfaces

Primary Dependency:
Application depends on product-peculiar functions

Secondary Dependency:
Application depends on product-peculiar qualities or configuration operations

Subject of this talk



Examples of Secondary Dependencies

Applies to: a Real-time ORB

Qualities

- Transports and protocols supported
- Reliability, or reliabilities available
- Efficiency of marshalling and de-marshalling event parameters
- Efficiency of de-multiplexing incoming method calls
- Buffer sizes, flow control, and buffer overflow handling
- Thread and thread priority utilization

Configuration Operations

- ORB initialization options
- ORB compilation options
- Implementation-provided configuration objects

Two Ways to Avoid Secondary Dependencies

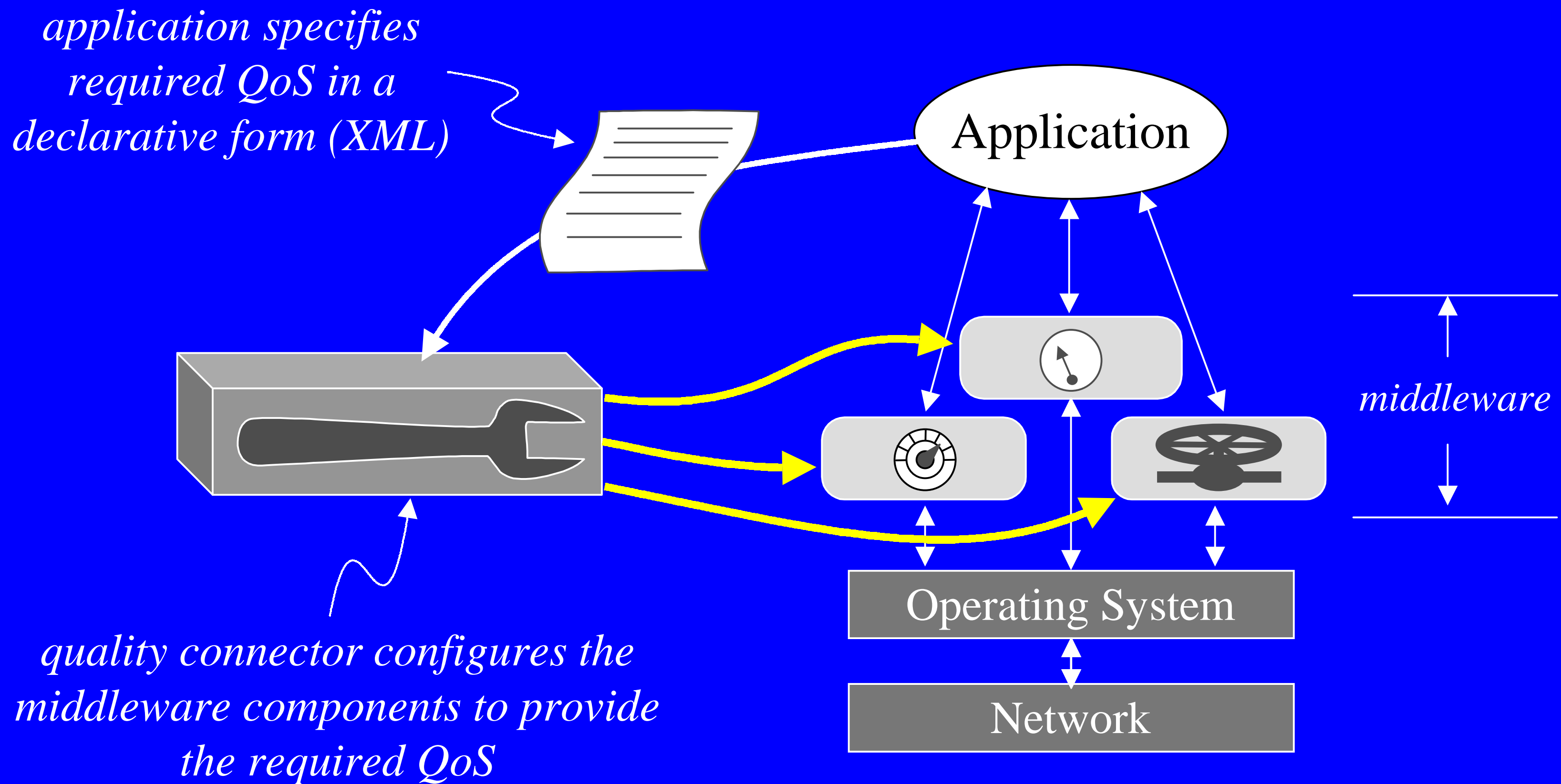
Standardize

- Standardize available configuration operations ← *not likely*
- Standardize offered qualities ← *generally not possible, not desirable*

Adapt

- Provide a software component, a <u>quality connector</u>, that
  – selects available qualities pursuant to application direction
  – performs required configuration operations
- Implementation of a quality connector is product-peculiar
- Interface to a quality connector could be standardized

A Quality Connector Acts on QoS Requirements Declarations



Data from Application to Quality Connector

The (declarative) data that the application passes to the quality connector consist of:

- Mode: a boolean combination of statements about the states of system components; the remaining data apply only when it's true
- Load: one or more constraints on the load that the application will impose
- QoS: one or more requirements on the QoSs that the application demands

```
<proposal>
    <mode>
        <or>
            <ci name="portSlingshot" state="onLine"/>
            <ci name="starboardSlingshot" state="onLine"/>
        </or>
    </mode>
    <QoS type="latency">
        <upperPoint secs="1.0" prob="0.99"/>
        <upperPoint secs="4.0" prob="0.9999"/>
    </QoS>
    <load type="interMessageTime">
        <upperPoint secs="1.0" prob="0.0001"/>
        <lowerPoint secs="1.0" prob="0.9999"/>
    </load>
    <load type="messageSize">
        <upperPoint bytes="256" prob="1.0"/>
        <upperPoint bytes="32"  prob="0.5"/>
    </load> <load type="priority">
        <urgency val="10"/>
        <importance val="2"/>
    </load>
</proposal>
```

proposal applies in this mode

there are QoS types other than latency -- e.g., jitter

this flow is periodic

priority determines how this request will compete with others for resources

Quality Connector Implementation

- In general, a quality connector may act at several points in the system life cycle:
  - when middleware is developed (e.g., options to the compiler that compiles the ORB)
  - when the application is written (weaver-style)
  - when the system is linked
  - when the system is initialized
  - during system execution
- We have built a prototype
  - acts only at run time
  - selects which of several physical channels will propagate events
- We will build a TAO-RT-event-configuring prototype

The Beautiful Vision

- Suppose the community of middleware users could agree on a core set of qualities of service and how to specify their values.

  E.g., a QoS for an event service is latency, and it is measured from the time any supplier's `push()` call begins until all consumers have received their `push()` calls. The 'latency' XML schema is at ….
- Vendors might be moved to supply quality connectors for their products. Users would be appreciative.
- Vendors might be moved to optimize their products using the standard QoS's as metrics.

BACKUP