Assessing Contemporary Modularization Techniques for Middleware Specialization

Akshay Dabholkar & Aniruddha Gokhale

aky@dre.vanderbilt.edu

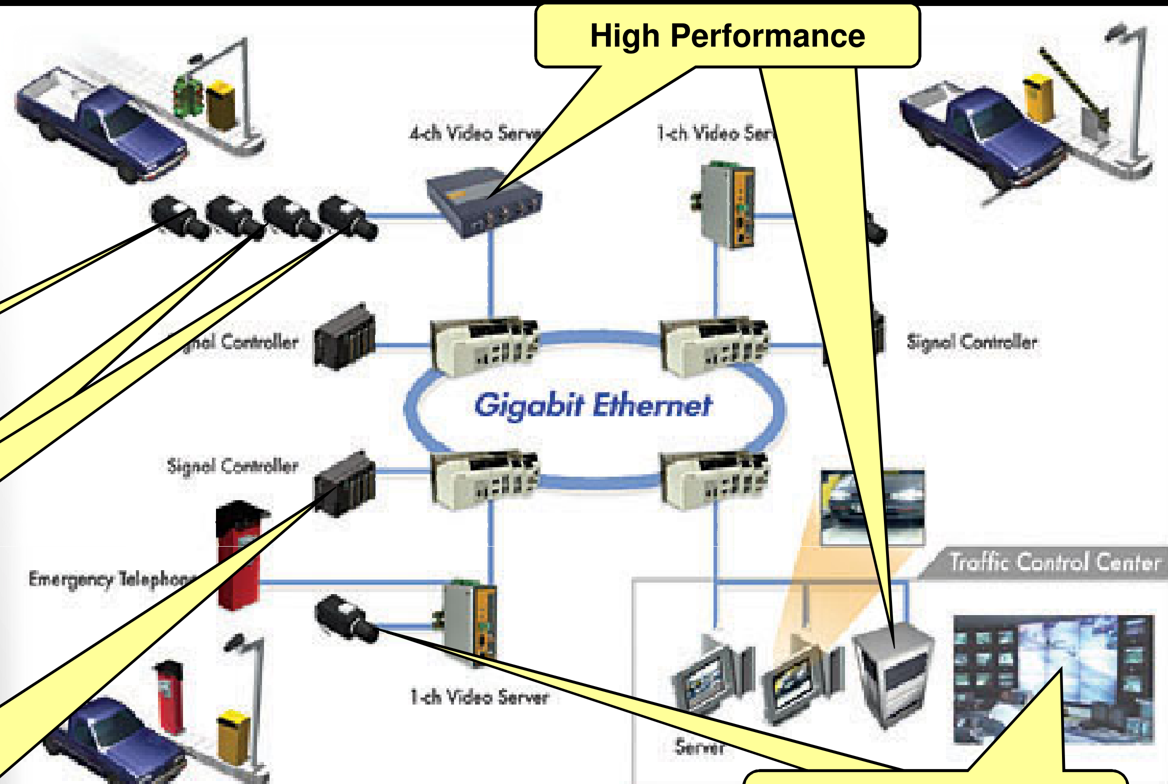
OOPSLA ACoM 2008 Workshop

Oct 19, 2008

Motivation: Intelligent Transportation Systems

- ***Dynamic*** and ***constantly evolving*** systems
- Rich in communication and information exchange
- ***Stringent QoS*** and performance requirements
- Invariably require middleware solutions and optimizations

Intelligent Transportation Systems: An example DRE system



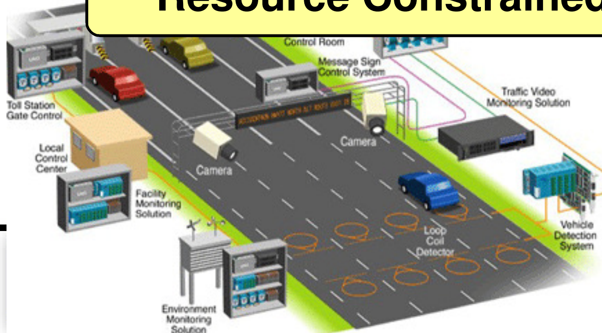
High Performance

Adaptive to weather

Differentiated Services

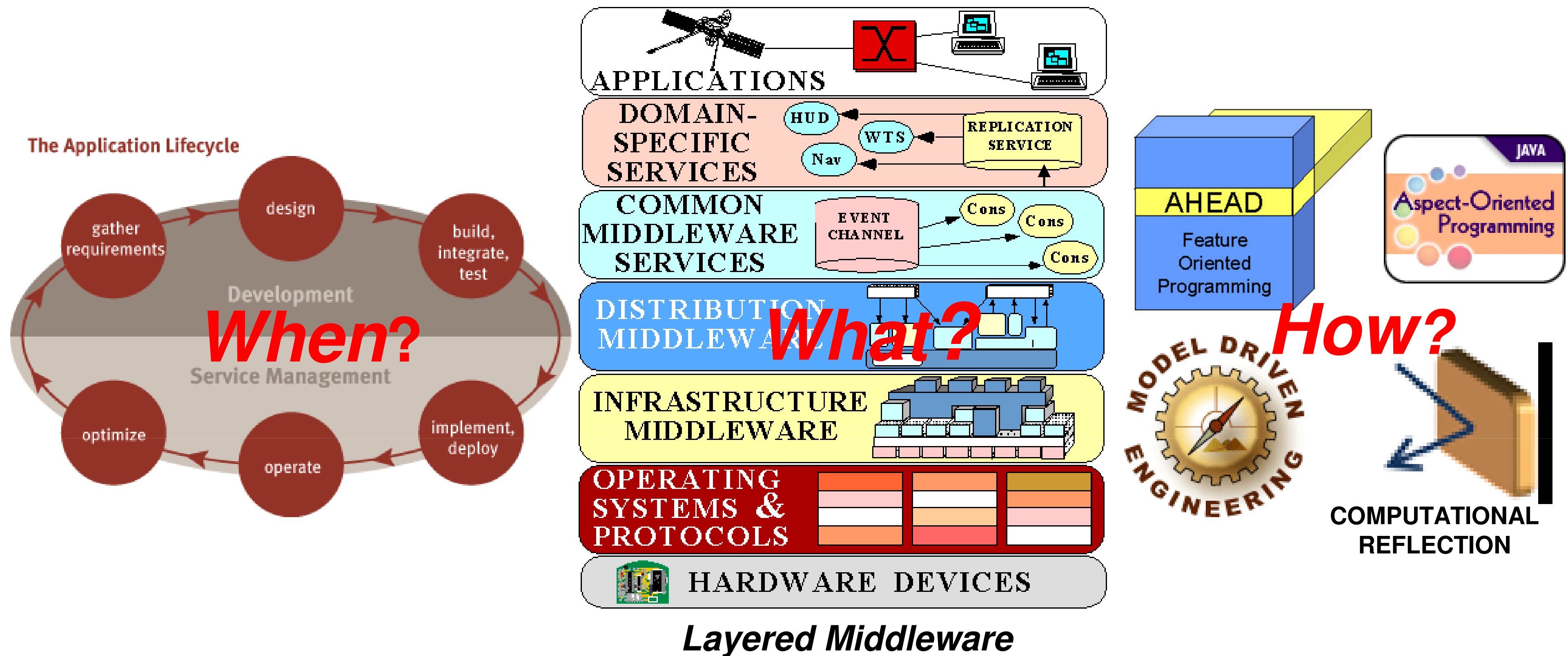
Real time

Resource Constrained



Gives rise to disparate middleware specialization needs *i.e.*, minimizing ***footprint,*** maximizing ***throughput, real-time*** computation, dynamic ***adaptation*** to unpredictable runtime changes

Challenges in Middleware Specialization



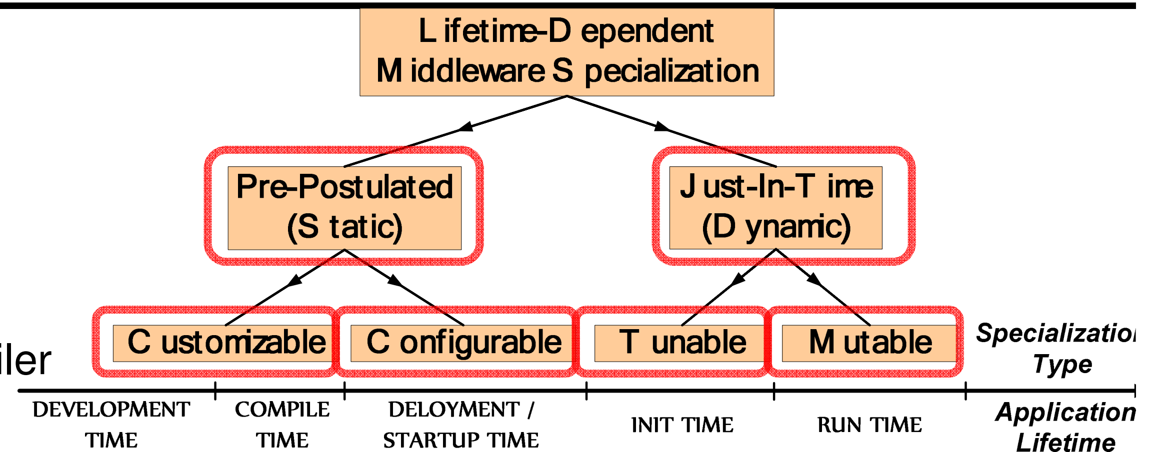
Layered Middleware

- Dynamic middleware-based systems require context specific *optimizations* and *adaptations i.e.,* **middleware specialization**
- ***How can we identify these specializations?***

Goal is to enable ***specializations*** at ***all stages*** of development life cycle in an ***integrated*** manner.

Challenge 1: When to specialize?

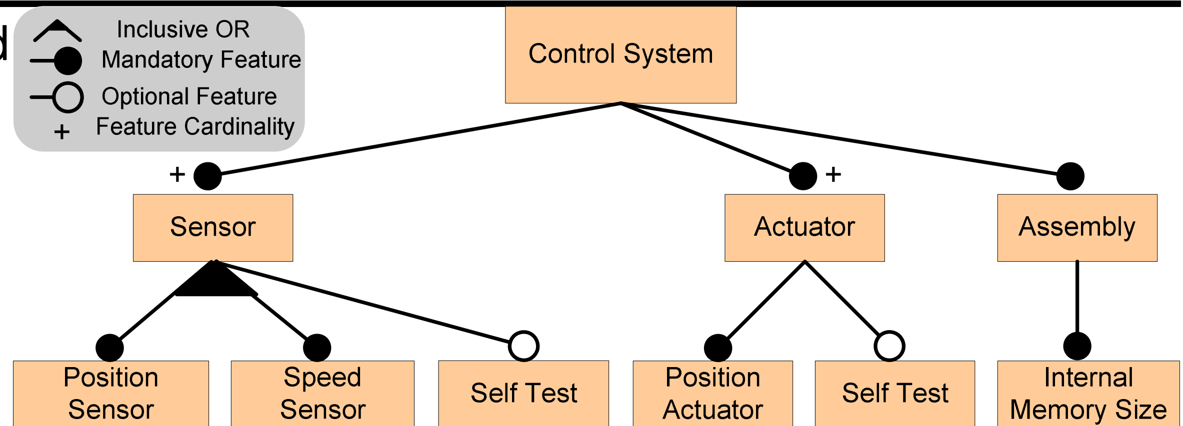
- **Pre-postulated**
  - **Customizable**
    - ***compile/link time***, ***after development time***
    - Generates specialized versions
    - *e.g.* static aspect weaving, compiler flags, precompiler directives,
  - **Configurable**
    - ***deployment/startup time***, ***after compile time***
    - *e.g.* CORBA PI, command-line parameters, ORB configuration files
- **Just-in-time (JIT)**
  - **Tunable**
    - ***Fixed middleware*** core
    - ***After*** *the* ***startup time*** but ***before run time – init / bootstrap time***
    - *e.g.* Component Configurator & Virtual Component patterns, two-step process (*compile time:* AOP, *run time:* Reflection)



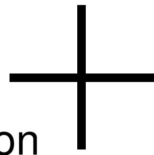
- **Mutable**
  - ***Most powerful (adaptive)***
  - ***No concept*** of ***fixed middleware*** core so can evolve into something completely different or ***unexpected***
  - *e.g. MetaSockets, Reflection, Late Composition, dynamic aspect weaving*

Challenge 2: What to specialize?

- *Context*: Resource constrained embedded software require footprint management
- *Solution:* Map application features to the supporting middleware features and optimize the middleware



- **Feature Augmentation**
  - Traditional middleware may not provide the desired supporting features
  - Add the required features
  - Commonly used by next generation middleware designed to overcome the limitations of monolithic architectures
  - Particularly useful to incorporate domain-specific semantics within middleware

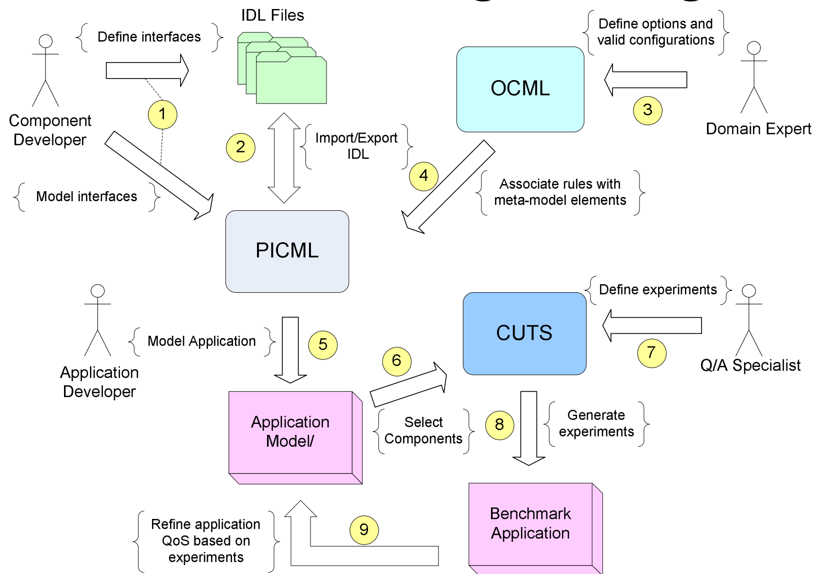


- **Feature Pruning**
  - Traditional middleware provides a broad range of features (***generic***) for multiple application domains
  - Remove unessential features
  - *Benefits:* reduces footprint, improves performance

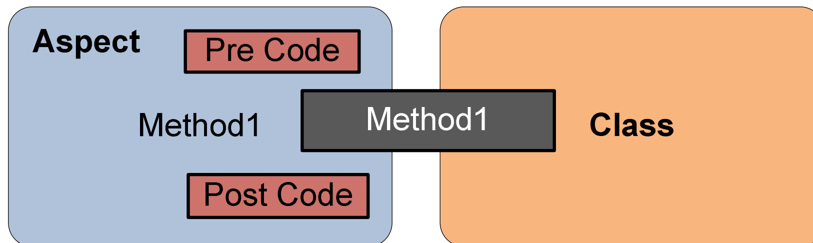
Need for ***consistent*** and ***correct*** feature management

Challenge 3: How to specialize?

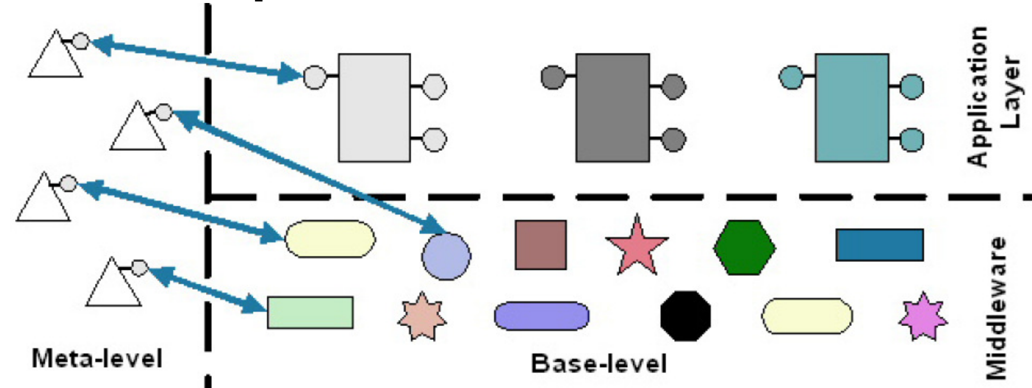
- **Model-Driven Engineering**



  - Integrates MDSD with QoS-enabled component middleware



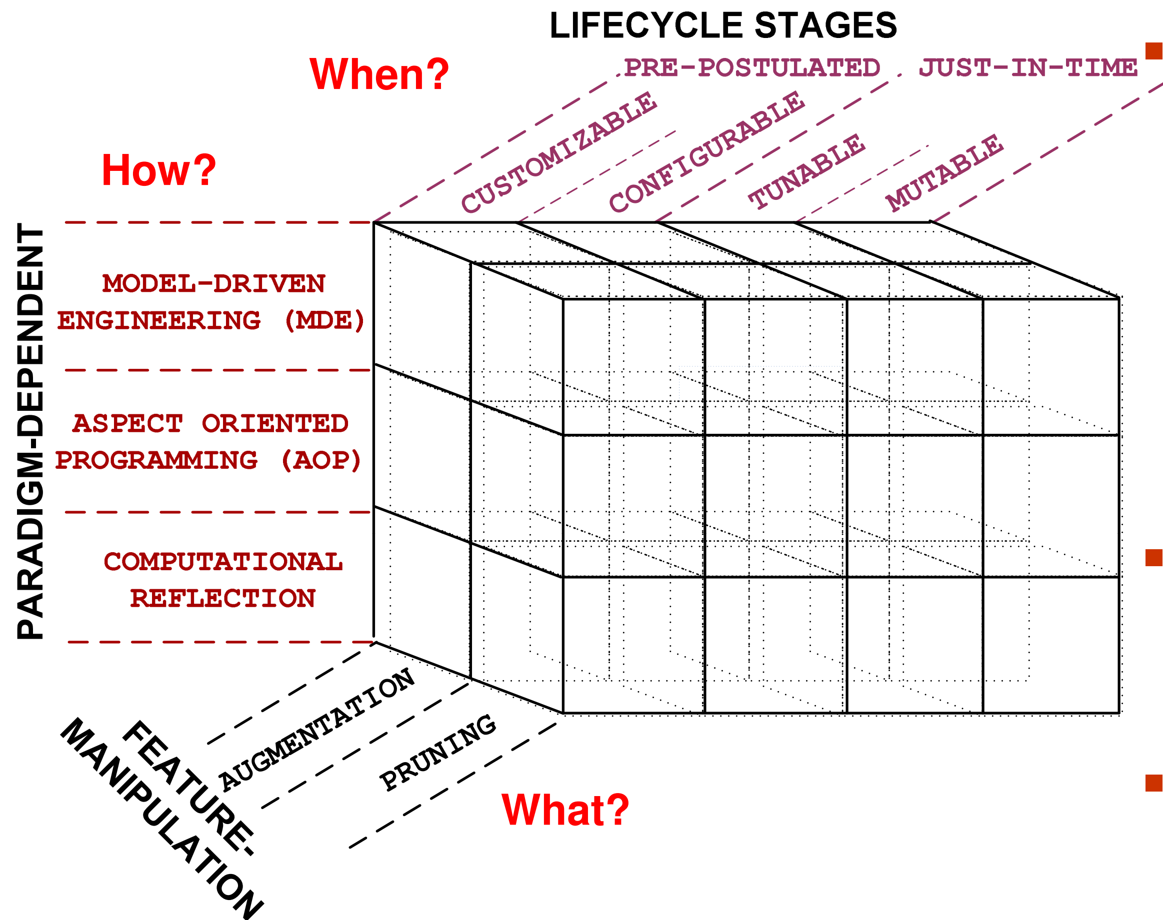
- **Computational Reflection**



  - Ability to introspect, reason about and adapt own behavior without exposing implementation

- **AOP for Middleware Specialization**
  - Factorization and separation of cross-cutting concerns from the middleware core
  - Generate customized versions of middleware for application-specific domains
  - *e.g.* CORBA Interceptors, Two-step process (*compile time:* AOP, *run time:* Reflection)

Taxonomy of Middleware Specialization Techniques



Three dimensional Taxonomy of Middleware Specializations

- Overlapping dimensions share concepts *e.g.* MDE/AOP includes both *feature pruning* & *augmentation* and can be used for *customization* as well as *tuning*
- Dimensions can be combined to produce new variants of specialization
- Serves as a ***guideline*** for synthesis of tools for design, V&V, analysis of specializations

Assessment of the Taxonomy

COMBINATION	USE CASES	STRENGTHS	WEAKNESSES	RELATED WORK
Pre-postulated + AOP	Weave/Prune at compile-time	Transparency without affecting core	Code Bloating	FACET, CLA, FOCUS, Bypassing Layers, AspectOpenORB
Pre-postulated + MDE	Weave/Prune only known features	Elegant design	Runtime specializations not possible	DTO, CLA, Modelware
Pre-postulated + Reflection	Inspect target platform features	Useful during deployment	Difficult to predict runtime conditions, preconceive hooks	AspectOpenORB, DTO
Just-in-time + AOP	Dynamic weaving of features	Dynamic Adaptation	Requires native platform support	JAsCo, PROSE, Abacus
Just-in-time + MDE	Self-healing/correcting systems	Validation of Specializations	Incur runtime overhead	Models@Runtime
Just-in-time + Reflection	Introspect runtime application features	Dynamic Adaptation & reconfiguration	Can cause unpredictable behavior, preconceive hooks	AspectOpenORB
AOP + FOP	ISD and SPLs	Better modularization of crosscutting features than AOP alone	Runtime specializations not possible, cause conflicts	AFMs, Caesar
FOP + MDE	SPLs	Better composition of features	Runtime specializations not possible, cause conflicts	FOMDD
AOP + Reflection	Composition based on application requirements	On-demand feature weaving	May cause conflicts	AspectOpenORB
AOP + MDE + FOP + Reflection	Design/Weave/Prune valid features combinations	Systematic, correct specialization process	Safe specializations is challenging	*Research Needed*

Open Middleware Specialization Research Areas

- Based on the survey at least four research areas show up:
  - Specialization of Domain-specific middleware services
    → ***Pattern Languages of Specializations***?
  - Mutable middleware specialization is powerful but a dangerous technique
    → ***Safe specializations?***
  - Application inconsistency caused by overlapping specialization techniques (similar to feature interaction problem in pattern recognition)
    → ***V&V of specializations?***
  - No single specialization that can adapt a entire distributed application
    → ***Integrated specializations?***

An **Integrated Tool Suite** based on **specialization pattern languages** to support ***synthesis, verification*** and ***validation*** of specializations

Thank You!

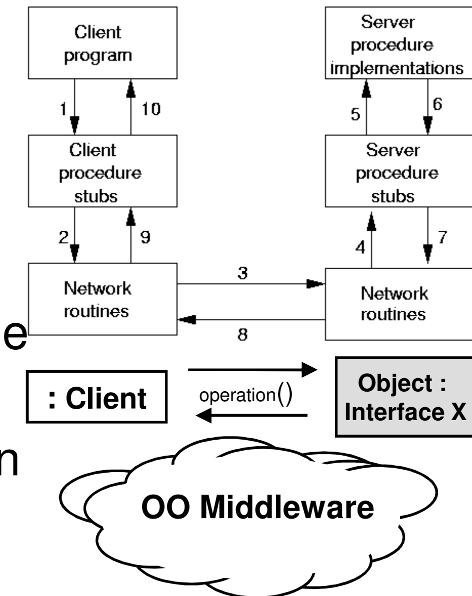
Questions?

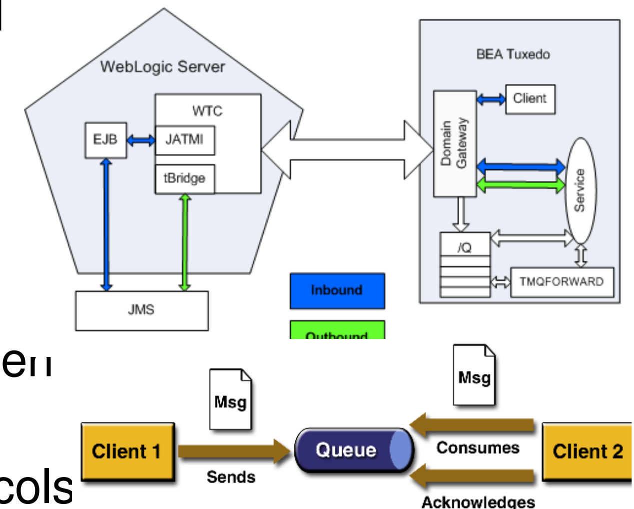
Summary of Surveyed Papers

SPECIALIZATION DIMENSION / SOLUTION APPROACHES	Lifetime	Feature	Paradigm
Aspects Beat Objects (Spinczyk et. al.)	Pre-postulated	BOTH	AOP
FACET (Cytron et. al.)	Pre-postulated	Augmentation	AOP
Modelware (Zhang et. al.)	Customizable	Augmentation	Design Patterns, AOP, MDE
Aspect Open-ORB (Batista et. al.)	Pre-postulated, Tunable	Augmentation	AOP, Computational Reflection
Deployment Time Optimization (Lee et. al.)	Pre-postulated	BOTH	MDE, Computational Reflection
Cross-Layer Specialization (Gregori et. al., Stockenberg et. al.)	Pre-postulated	BOTH	Design Patterns, AOP, MDE
Bypassing Layers (Devambu et. al.)	Pre-postulated, Mutable	BOTH	AOP

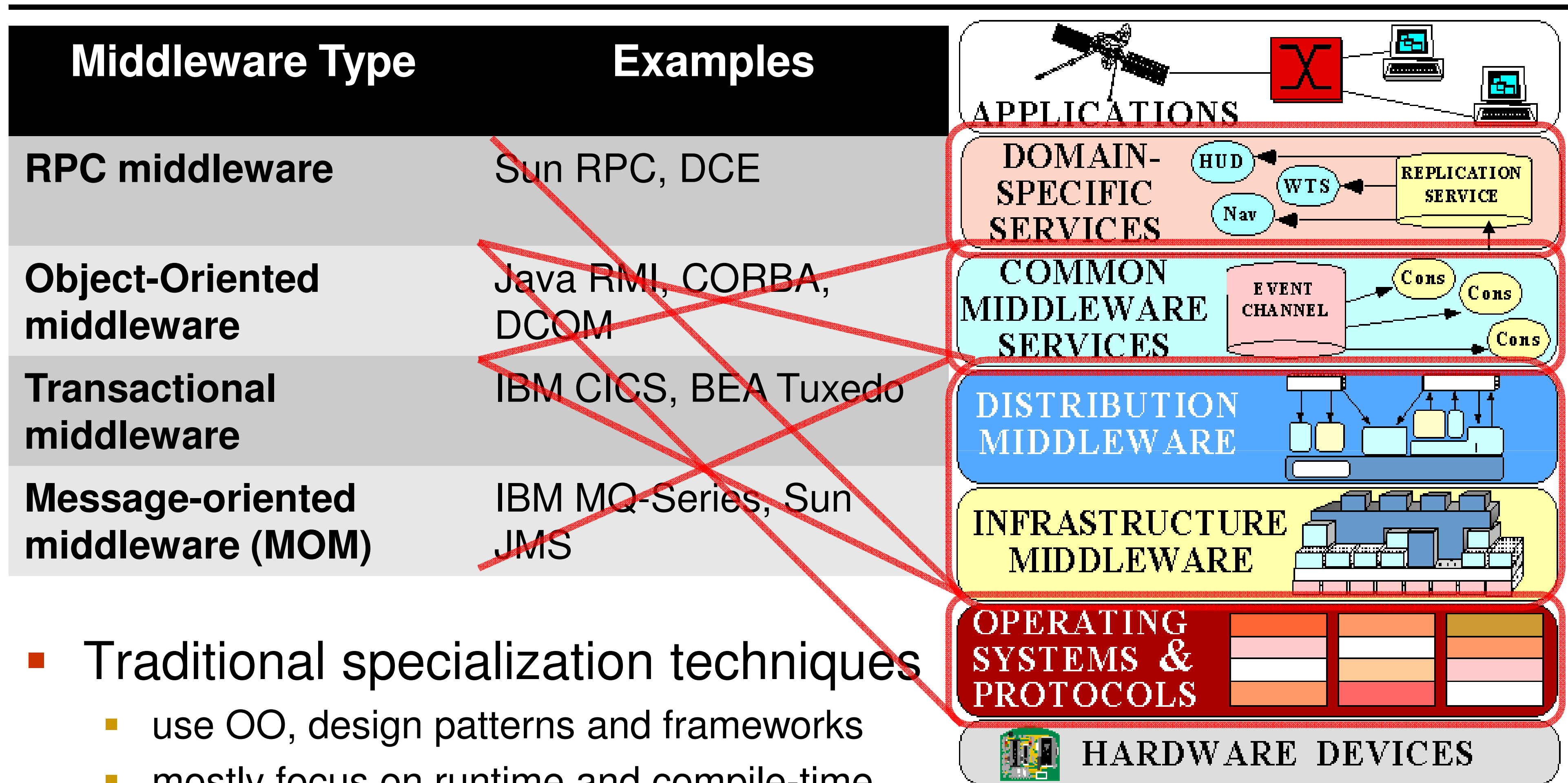
Traditional Middleware

- RPC Middleware
  - Extends traditional procedure calls to remote
  - Specialized using dynamic binding selection
- Object Oriented Middleware
  - combines object-oriented programming paradigm and the RPC architecture
  - Specialized using design patterns, frameworks, reflection
- Transactional Middleware
  - supports distributed transactions among distributed processes
  - Specialized similar to OO middleware using most resource and performance efficient bindings
- Message Oriented Middleware (MOM)
  - facilitates asynchronous message exchange between clients and servers using the message-queue
  - Specialized for a particular routing substrate, protocols





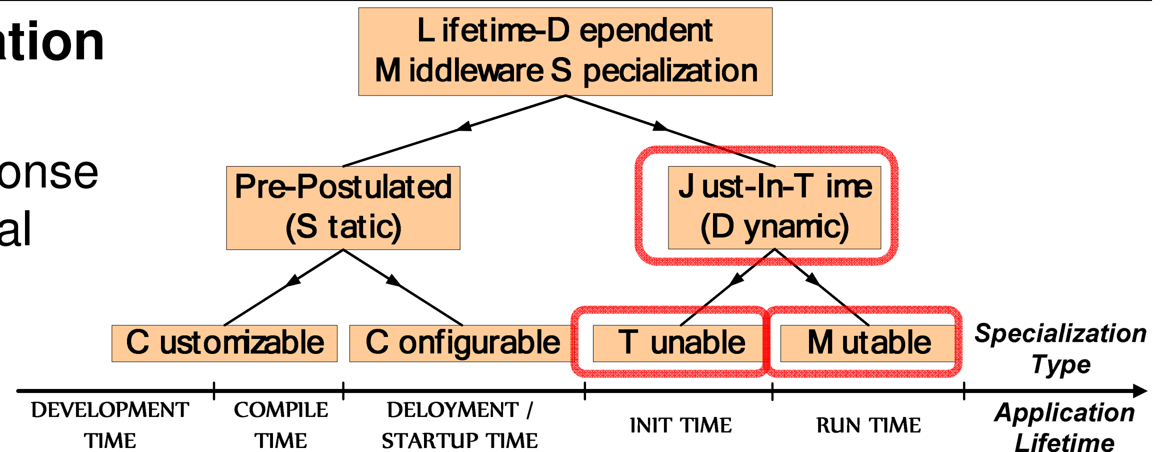
Traditional Middleware Specialization



Middleware Type	Examples
RPC middleware	Sun RPC, DCE
Object-Oriented middleware	Java RMI, CORBA, DCOM
Transactional middleware	IBM CICS, BEA Tuxedo
Message-oriented middleware (MOM)	IBM MQ-Series, Sun JMS

- Traditional specialization techniques
  - use OO, design patterns and frameworks
  - mostly focus on runtime and compile-time
  - No way to integrate multiple specializations
  - don’t focus on domain-specific specializations

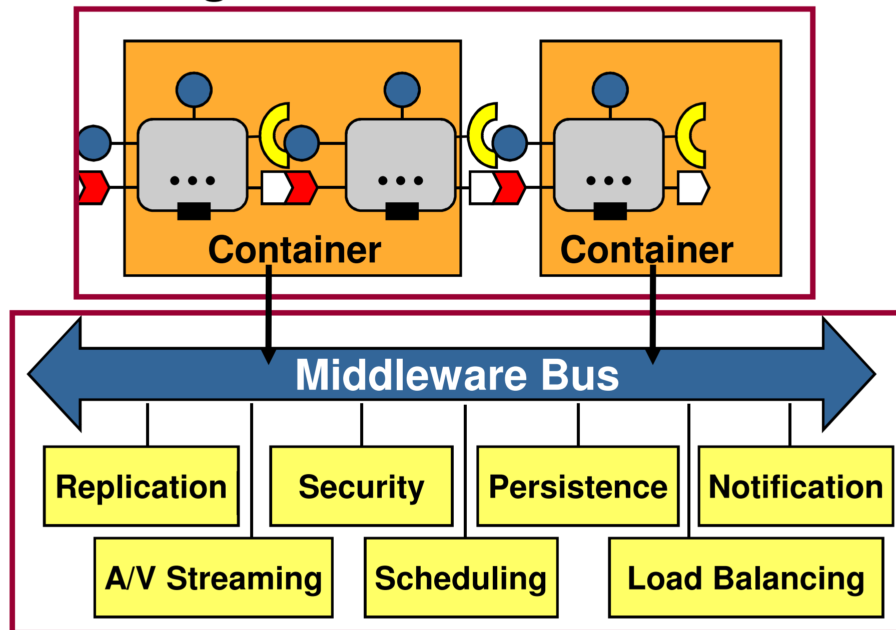
Challenge 2: When to specialize? (2/2)



- **Just-in-time (JIT) Specialization**
  - Identifies the requirements of ***running applications*** in response to functional and environmental changes
  - **Tunable**
    - ***Fixed middleware*** core
    - ***After*** *the* ***startup time*** but ***before run time – init / bootstrap time***
    - *e.g.* Component Configurator & Virtual Component patterns, two-step process (*compile time:* AOP, *run time:* Reflection)
  - **Mutable**
    - ***Most powerful (adaptive)***
    - ***No concept*** of ***fixed middleware*** core so can evolve into something completely different or ***unexpected***
    - *e.g. MetaSockets, Reflection, Late Composition, dynamic aspect weaving*

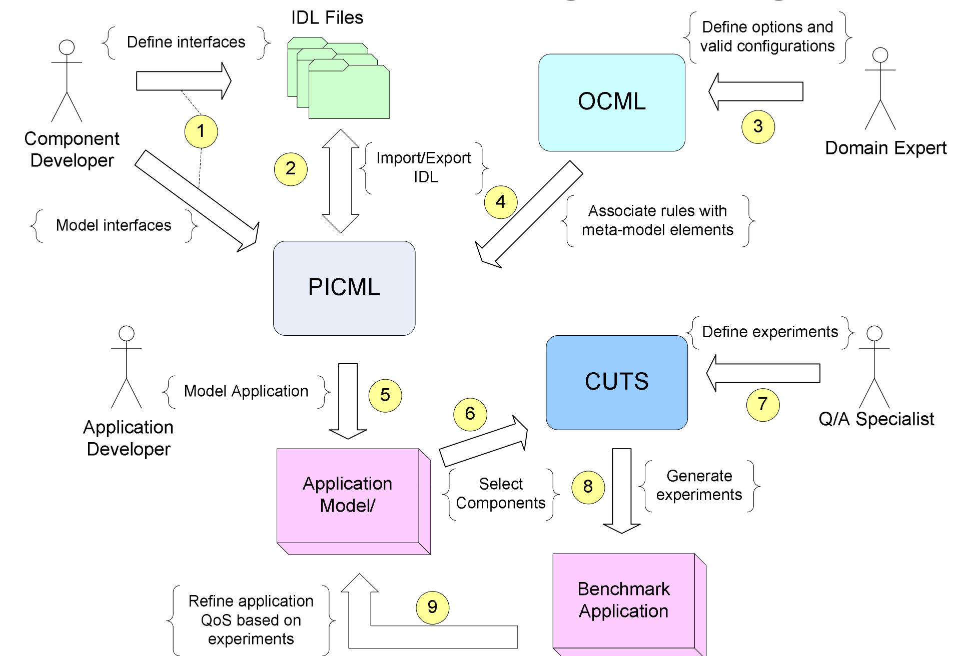
Challenge 3: How to specialize? (3/3)

- **Component-Based Middleware Design**



- *Reusable*, *self contained* software units of composition *independently* developed, composed, configured and deployed
- Provides a *flexible* and *extensible* middleware system to aid specializations
- Dynamically *adapted* and *evolved* using late composition

- **Model-Driven Engineering**



- Integrates MDSD with QoS-enabled component middleware
- Provides a *systematic process* reasoning about system design, composition, packaging, and deployment
- Facilitates analysis, emulation, verification, validation and feedback.

Middleware Specialization Solutions (1/2)

- *AOP + Feature management*

  Footprint minimization for embedded systems product lines

  Aspects Beat Objects (Spinczyk et. al.)
  - AOP as an alternative to OO
  - AOP achieves better SoC than OO with significantly smaller memory footprints
  - ***Features Pruning***

  FACET (Cytron et. al.)
  - Minimal middleware core
  - Encapsulate crosscutting concerns into user selectable aspects
  - ***Feature Augmentation***
- *AOP + Reflection*

  Aspect OpenORB (Batista et. al.)
  - Separation of application code and configuration files
  - ***Feature Augmentation*** though AOP
  - ***Dynamic specialization*** though MOP
- *AOP + MDA*

  Modelware (Zhang et. al.)
  - "Intrinsic " or middleware core or base view –
    - essential, invariant, architectural elements
    - composed with ***design patterns***
  - "Extrinsic" or aspect view –
    - optional elements subject to refinements
    - domain variations
    - ***Feature Augmentation***
  - Concrete middleware instances
    - **Realization** – select abstraction implementations from both views
    - **Projection** – create ontological relationships between both views

Middleware Specialization Solutions (2/2)

- *MDE + Reflection + Pre-postulated*

  Deployment Time Optimization (Lee et. al.)

  IBM BluePencil

  - Based on configuration settings of target environment
  - Select correct binding (Java RMI vs. SOAP)
  - Select drivers offering only the required features and better performance (databases)

- *AOP + Pre-postulated + Mutable*

  Bypassing Middleware Layers (Devambu et. al.)

  - Eliminates rigid middleware layer processing
  - Beneficial where response is a simple function of the request input parameters
  - Uses AOP and code generation (IDL-based) to build bypassing implementations

- *Patterns + AOP + MDE*

  Cross-Layer Specialization

  - *Explicit cooperation* between layers *e.g.* Gnutella (Gregori et. al.)
  - *Functionality migration* to lower layers *e.g.* Vertical Migration (Stockenberg et. al.)
  - *Generate context dependent versions* of system calls using partial evaluation
  - Incorporate routing, caching in networking layers *e.g.* Web servers
  - *Manual/automated bundling* of several system calls to do more work with fewer protection boundary crossings