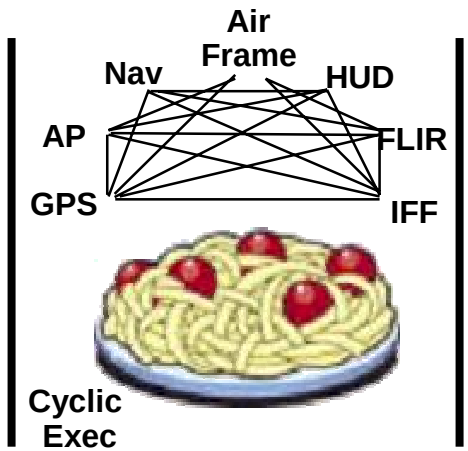
QUICKER: A Model-driven QoS Mapping Tool for QoS-enabled Component Middleware

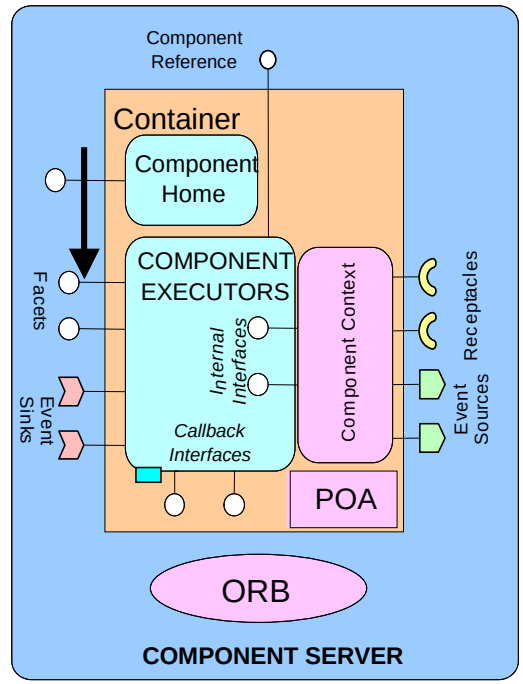
Amogh Kavimandan, Krishnakumar Balasubramanian, Nishanth Shankaran, Aniruddha Gokhale, & Douglas C. Schmidt
amoghk@dre.vanderbilt.edu

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee

Context

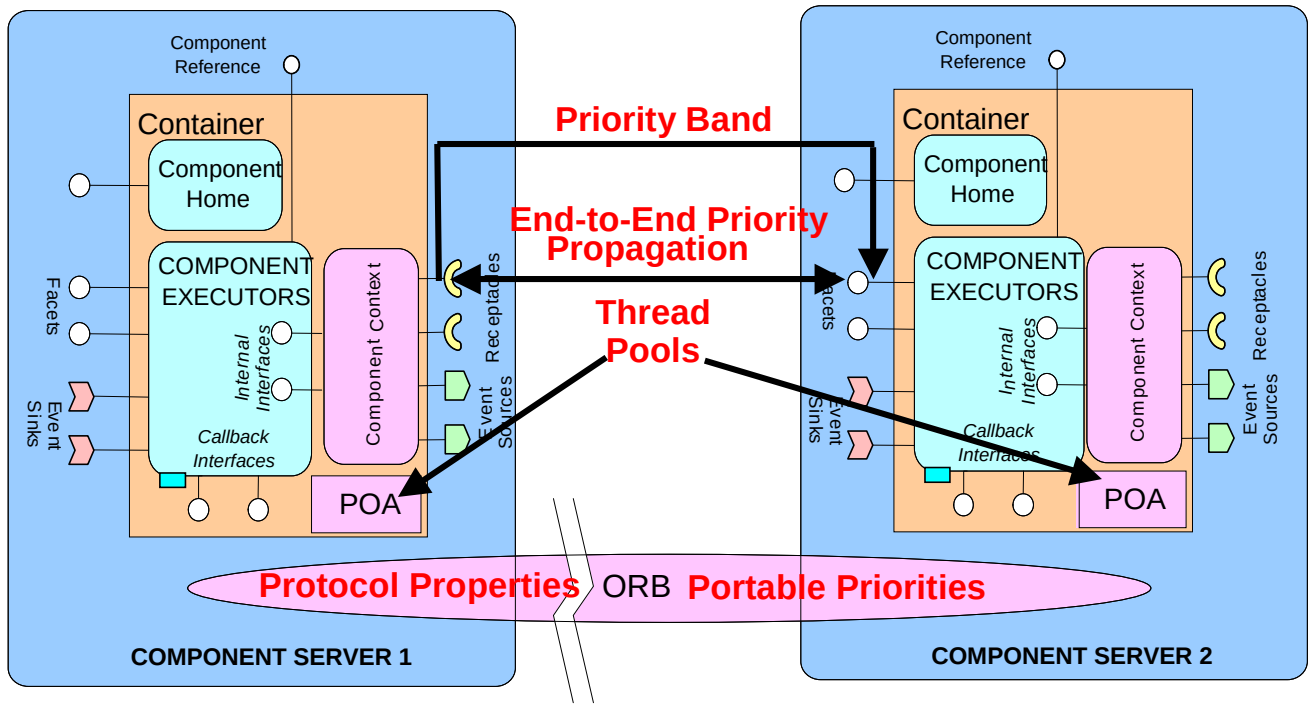
- Benefits of QoS-enabled middleware technologies
  - Raise the level of abstraction





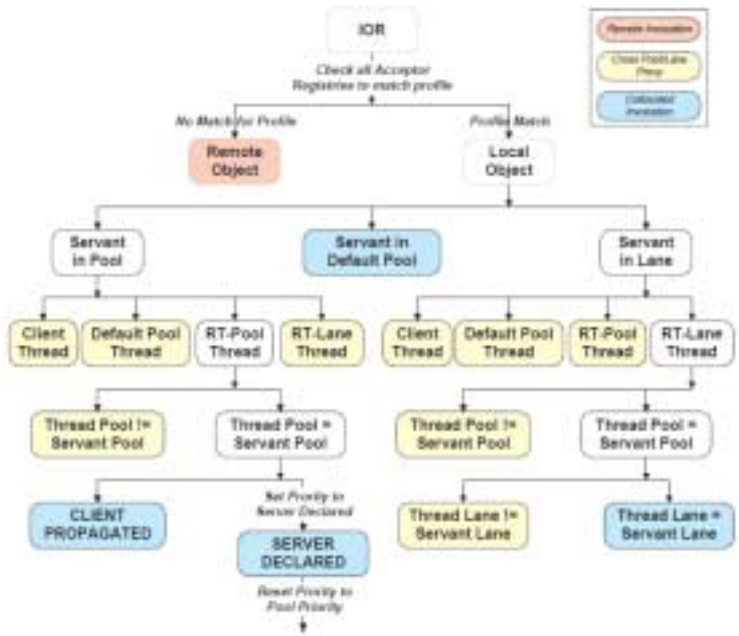
Context

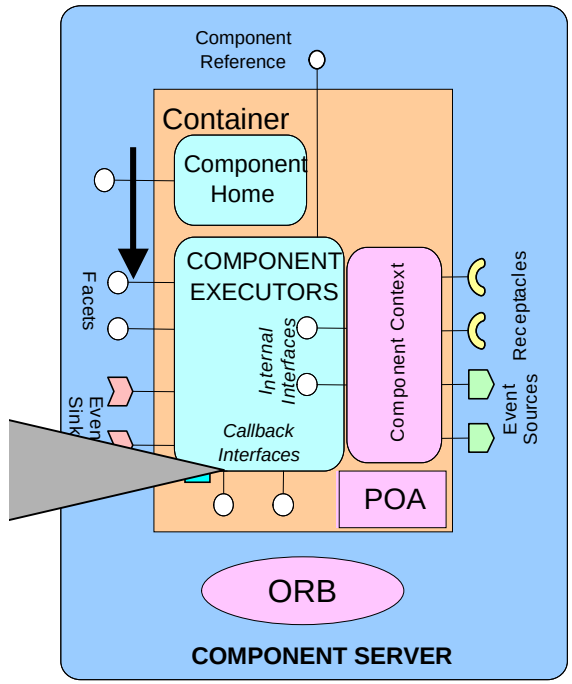
- Benefits of QoS-enabled middleware technologies
  - Raise the level of abstraction
  - Support many quality of service (QoS) configuration knobs



Context

- **Benefits** of QoS-enabled middleware technologies
  - Raise the level of abstraction
  - Support many quality of service (QoS) configuration knobs



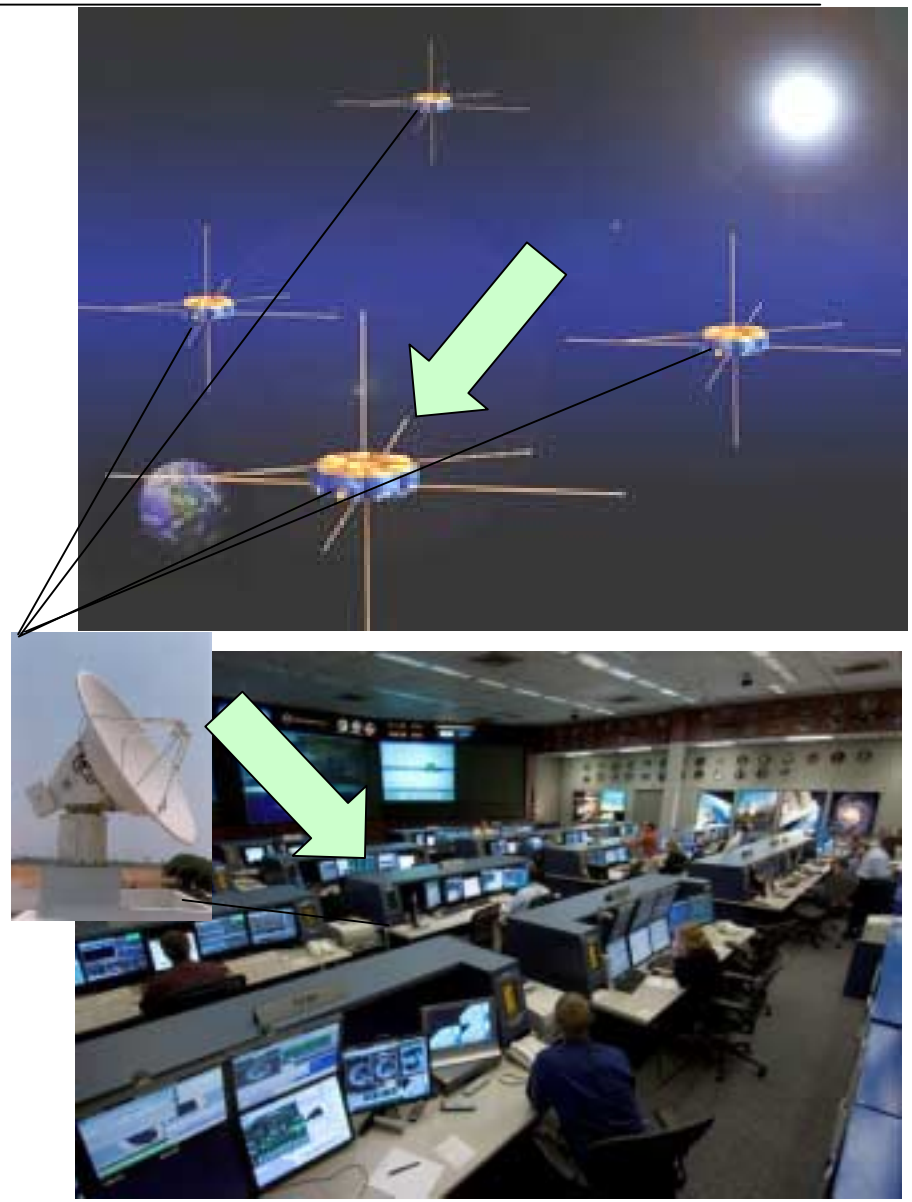


- **Drawbacks** of QoS-enabled middleware technologies
  - Achieving desired QoS increasingly a system *QoS configuration* problem, not just an initial system *functional design* problem

Lack of effective QoS configuration tools result in QoS policy mis-configurartions that are hard to analyze & debug

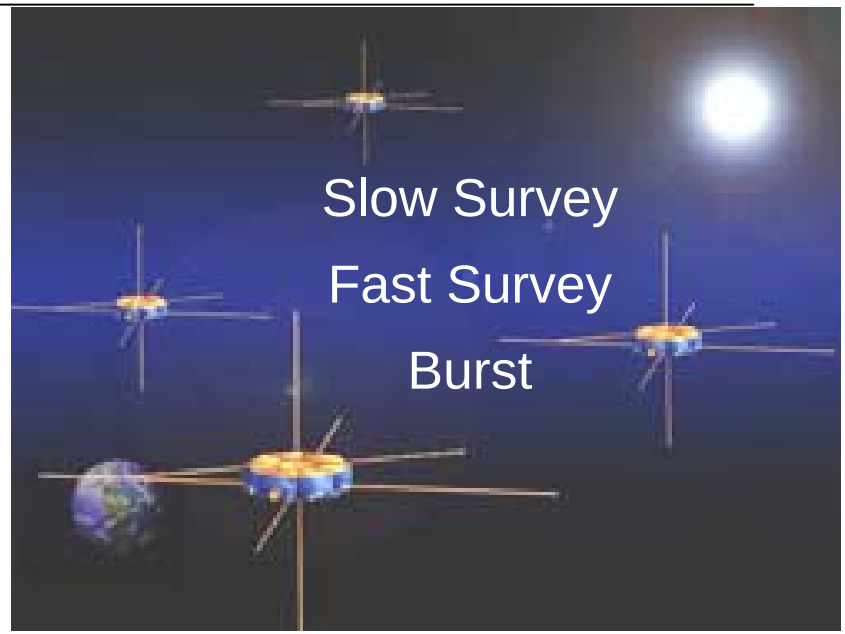
Motivating Application: NASA MMS Mission

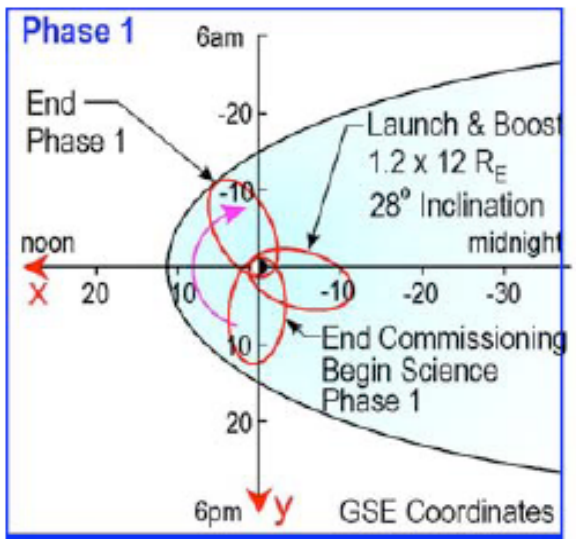
- NASA's Magnetospheric MultiScale (MMS) space mission consists of four identically instrumented spacecraft & a ground control system
  - Collect mission data
  - Send it to ground control at appropriate time instances

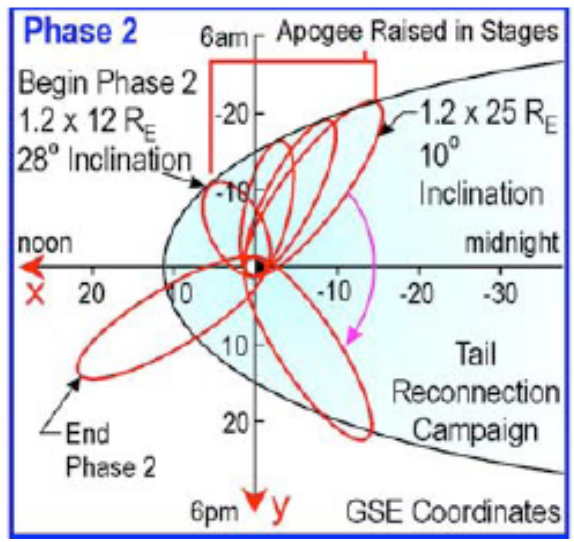


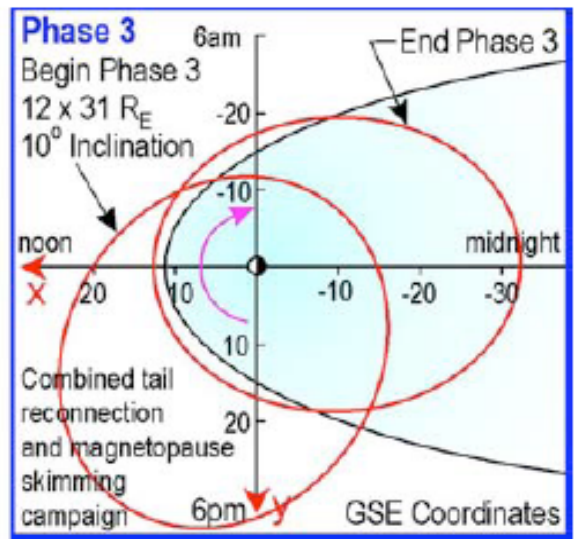
Motivating Application: NASA MMS Mission

- MMS mission QoS requirements span two dimensions
  - Multiple modes of operation
  - Varying importance of data collection activity of satellite sensors based on mission phase



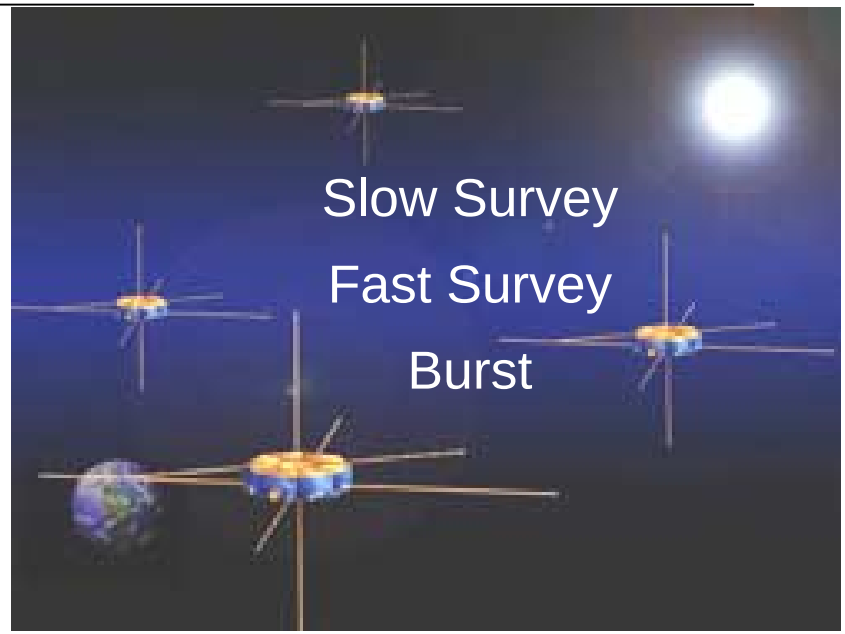


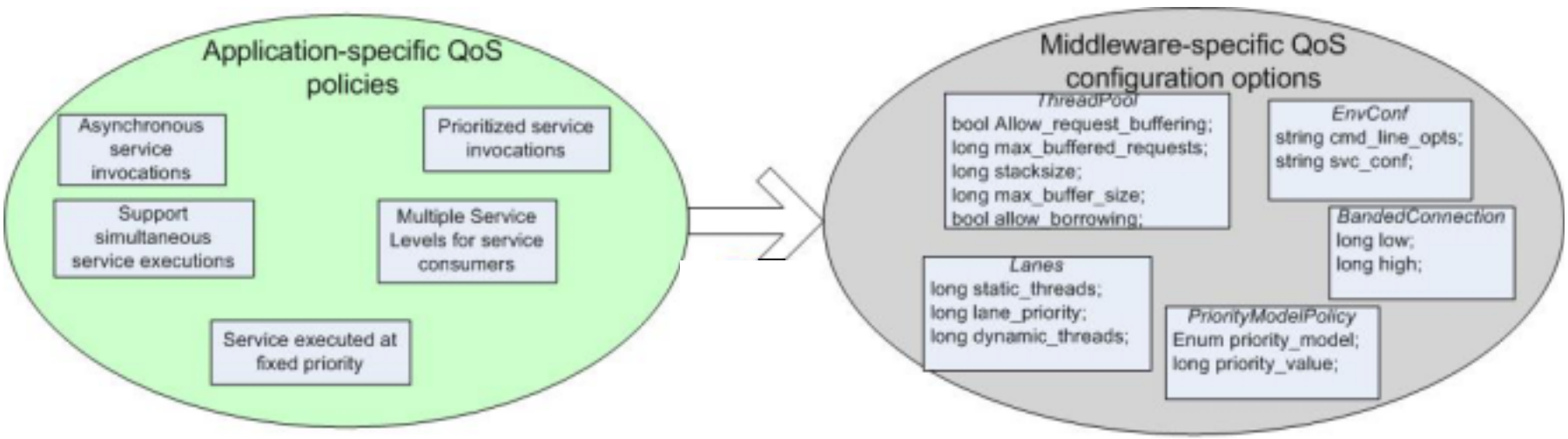




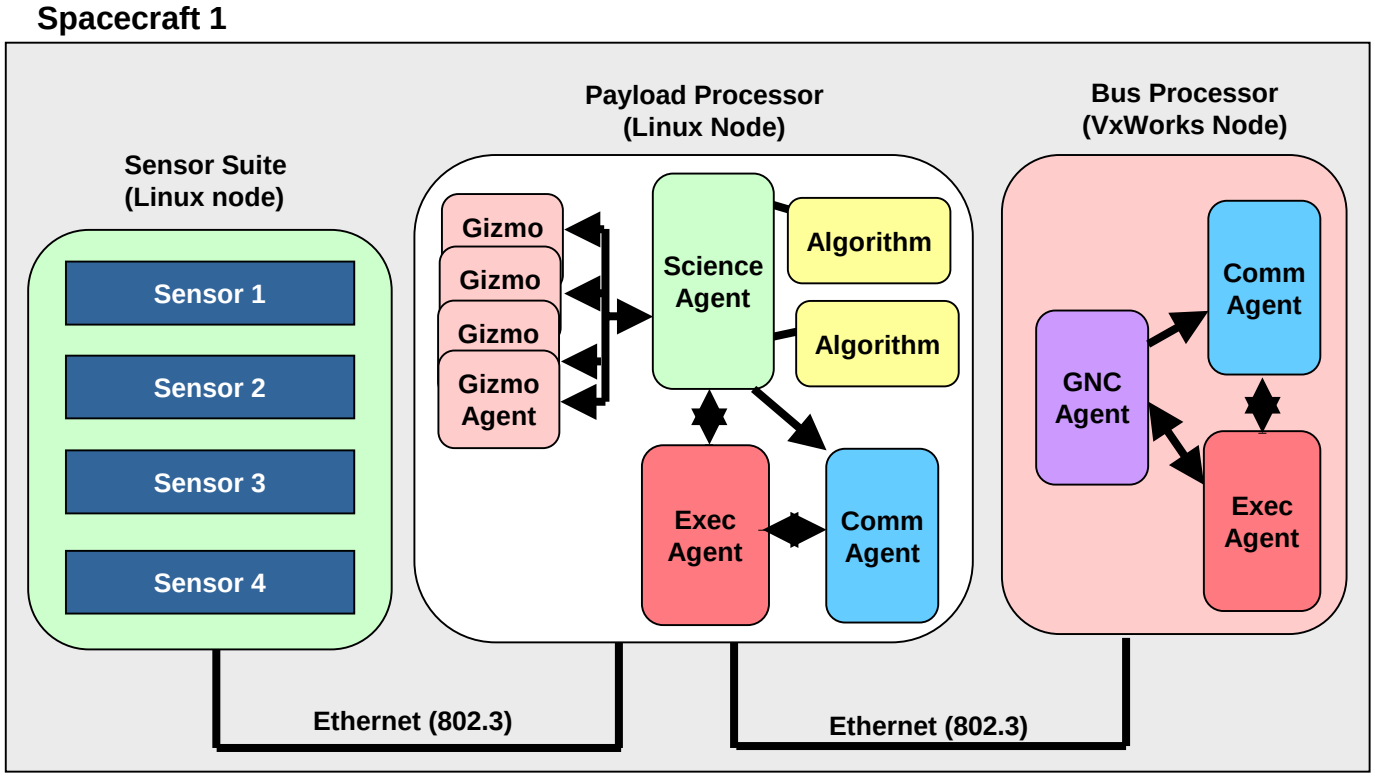
Motivating Application: NASA MMS Mission

- MMS mission QoS requirements span two dimensions
  - Multiple modes of operation
  - Varying importance of data collection activity of satellite sensors based on mission phase
- Need to translate QoS policies into QoS configuration options & resolve QoS dependencies





Component-based MMS Mission Prototype

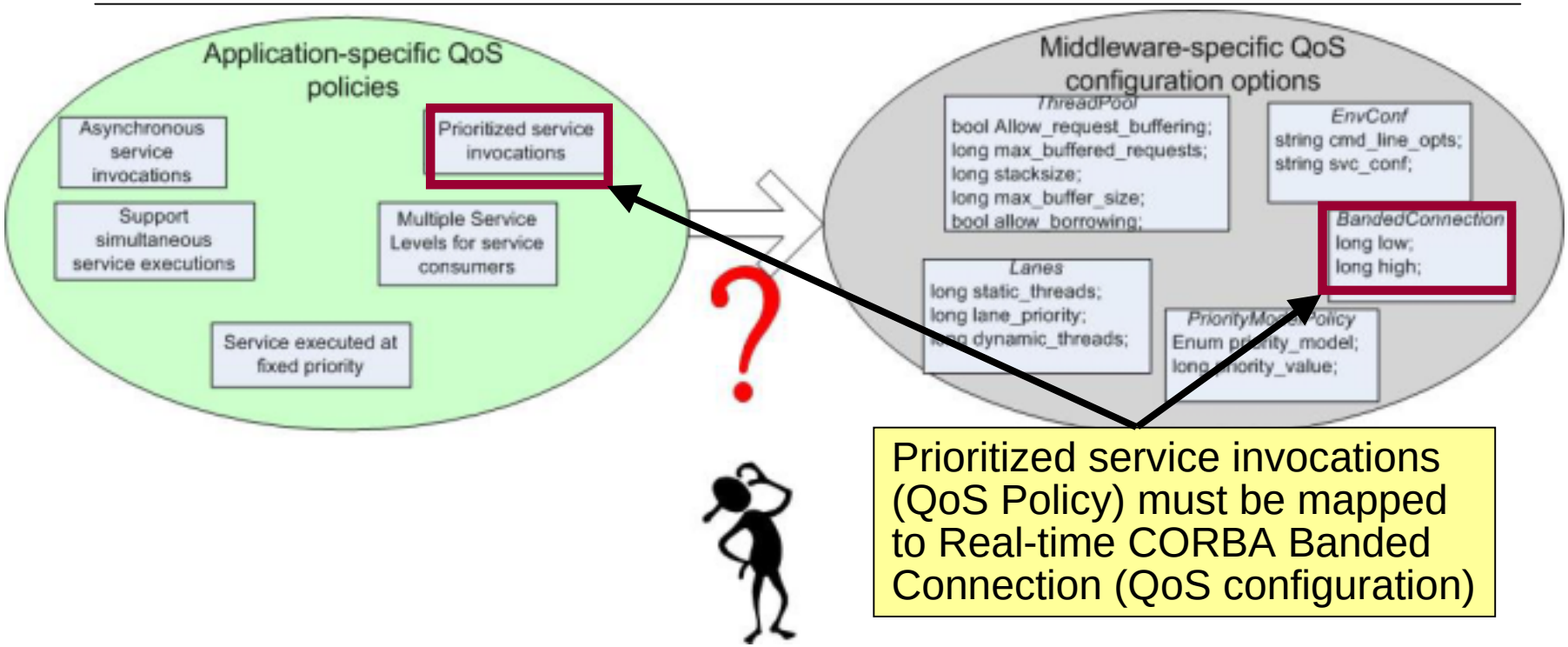


- MMS application components are bundled together into *hierarchical assemblies*
- Assembly package metadata conveys component interconnections & implementation alternatives

- Several different assemblies tailored to deliver different end-to-end QoS behaviors and/or algorithms can be part of the package
  - e.g., full-scale MMS has 100's of components & 10's of assemblies
- Packages describing the components & assemblies can be scripted via XML descriptors generated by automated model-driven tools

MMS prototype developed using Component-Integrated ACE ORB (CIAO)

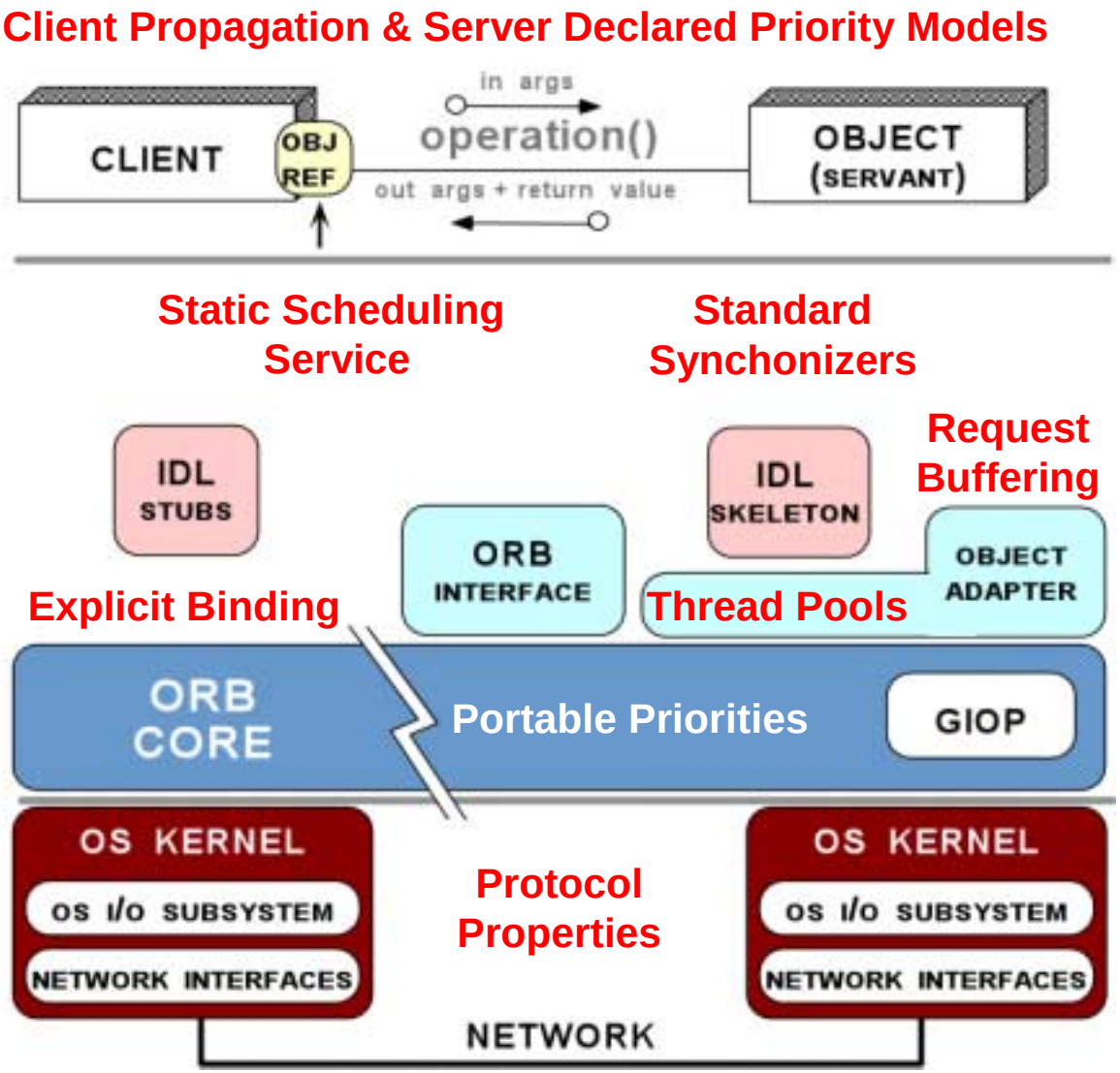
Challenge 1: Translating QoS Policies to QoS Options



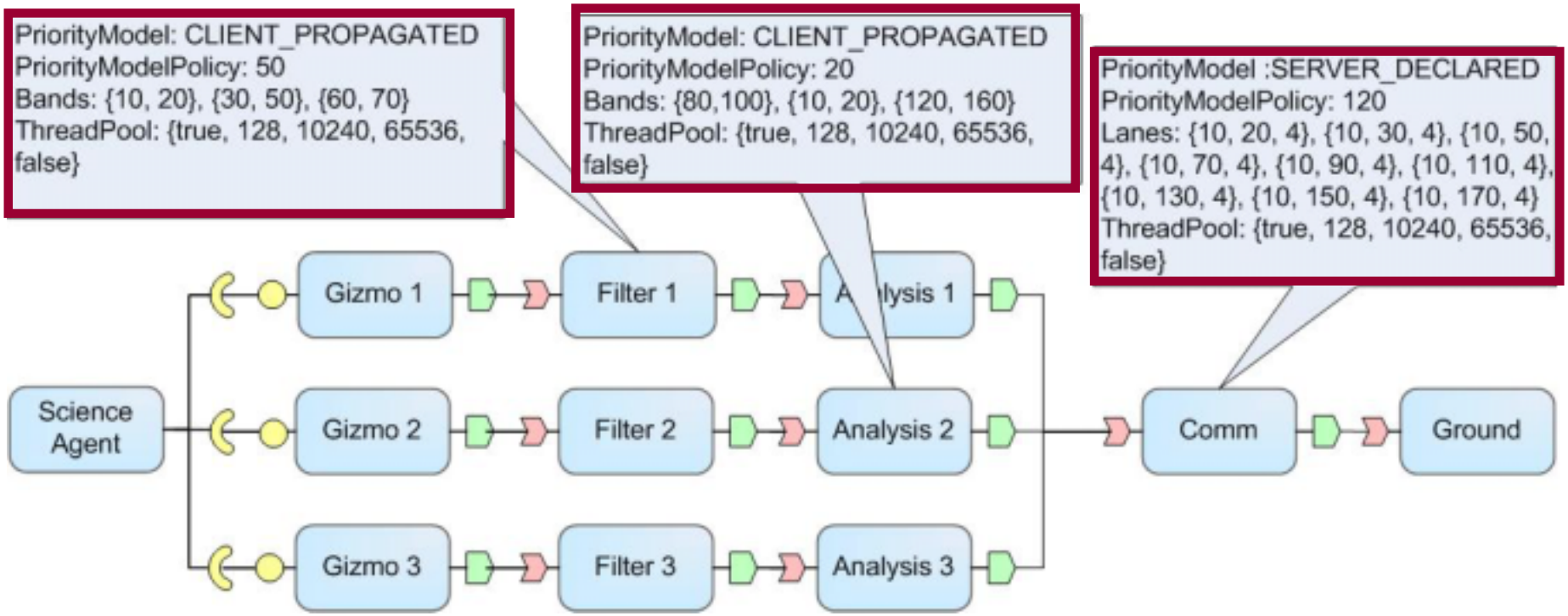
- Large gap between application QoS policies & middleware QoS configuration options
  - Bridging this gap is necessary to realize the desired QoS policies
- The mapping between application-specific QoS policies & middleware-specific QoS configuration options is non-trivial, particular for large systems

Challenge 1: Translating QoS Policies to QoS Options

- Conventional mapping approach requires deep understanding of the middleware configuration space
  - e.g., multiple types/levels of QoS policies require configuring appropriate number of thread pools, threadpool lanes (server) & banded connections (client)

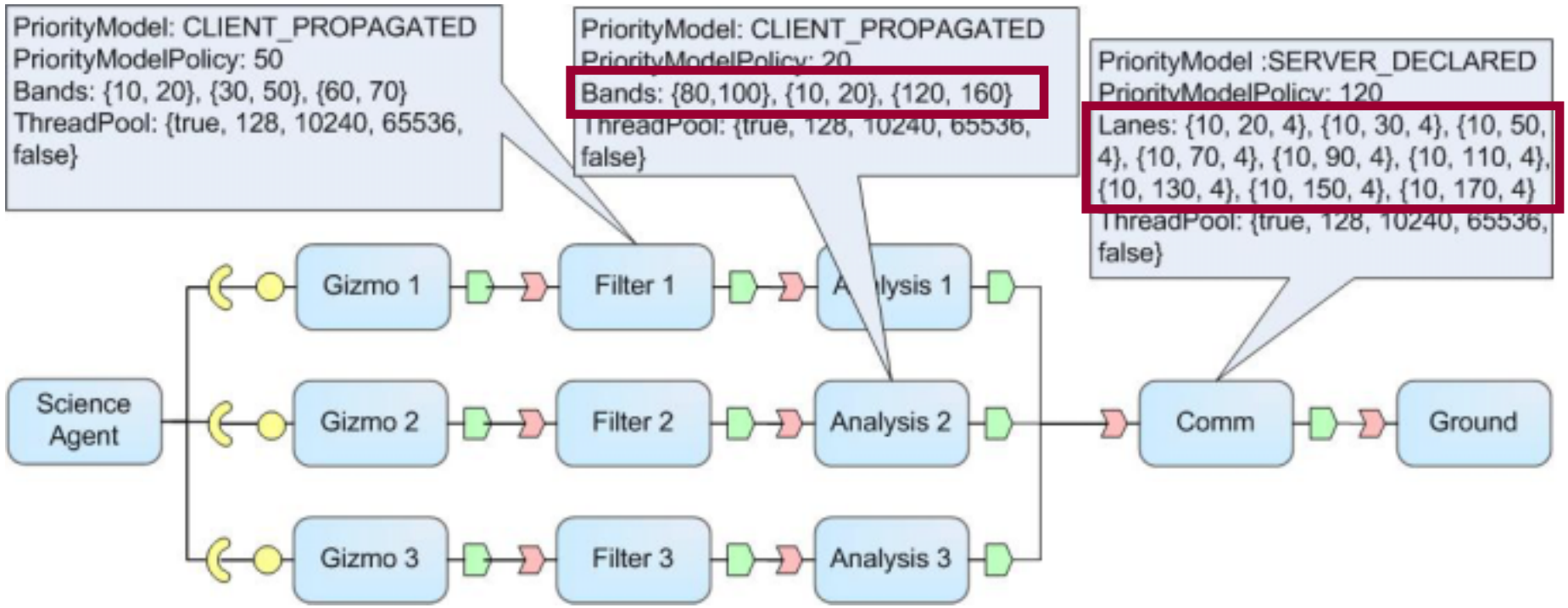


Challenge 2: Choosing Appropriate QoS Option Values



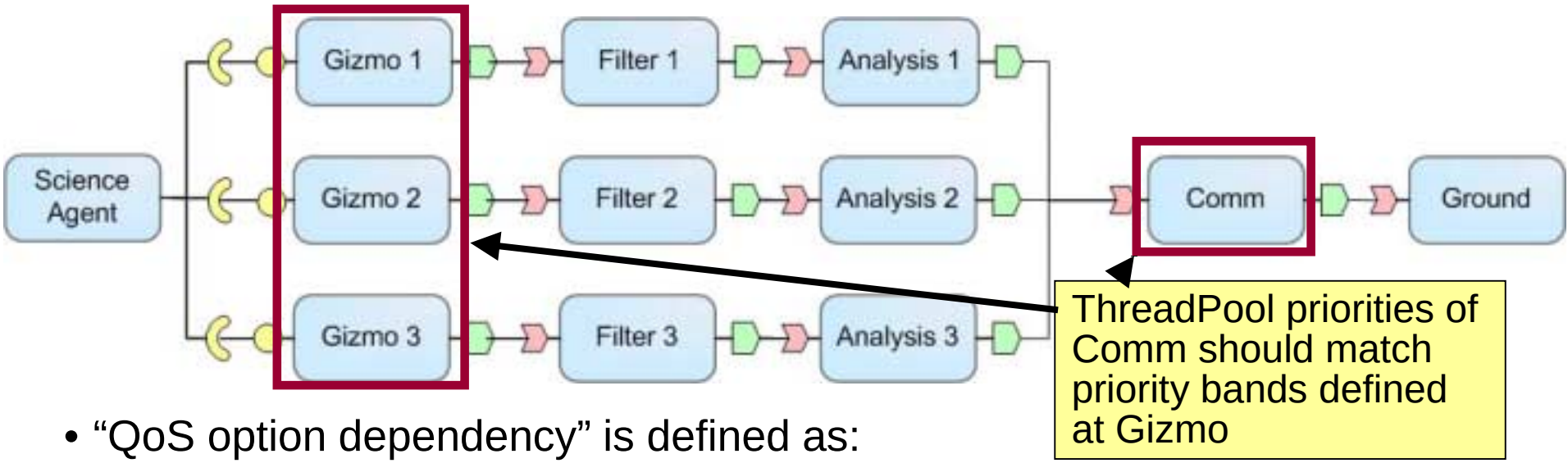
- Individually configuring component QoS options is tedious & error-prone
  - e.g., ~10 distinct QoS options per component & ~140 total QoS options for entire NASA MMS mission prototype
- Manually choosing valid values for QoS options does not scale as size & complexity of applications increase

Challenge 3: Validating QoS Options



- Each QoS option value chosen should be validated
  - e.g., Filter priority model is CLIENT_PROPAGATED, whereas Comm priority model is SERVER_DECLARED
- Each system reconfiguration (at design time) should be validated
  - e.g., reconfiguration of *bands* of Analysis should be validated such that the modified value corresponds to (some) *lane* priority of the Comm

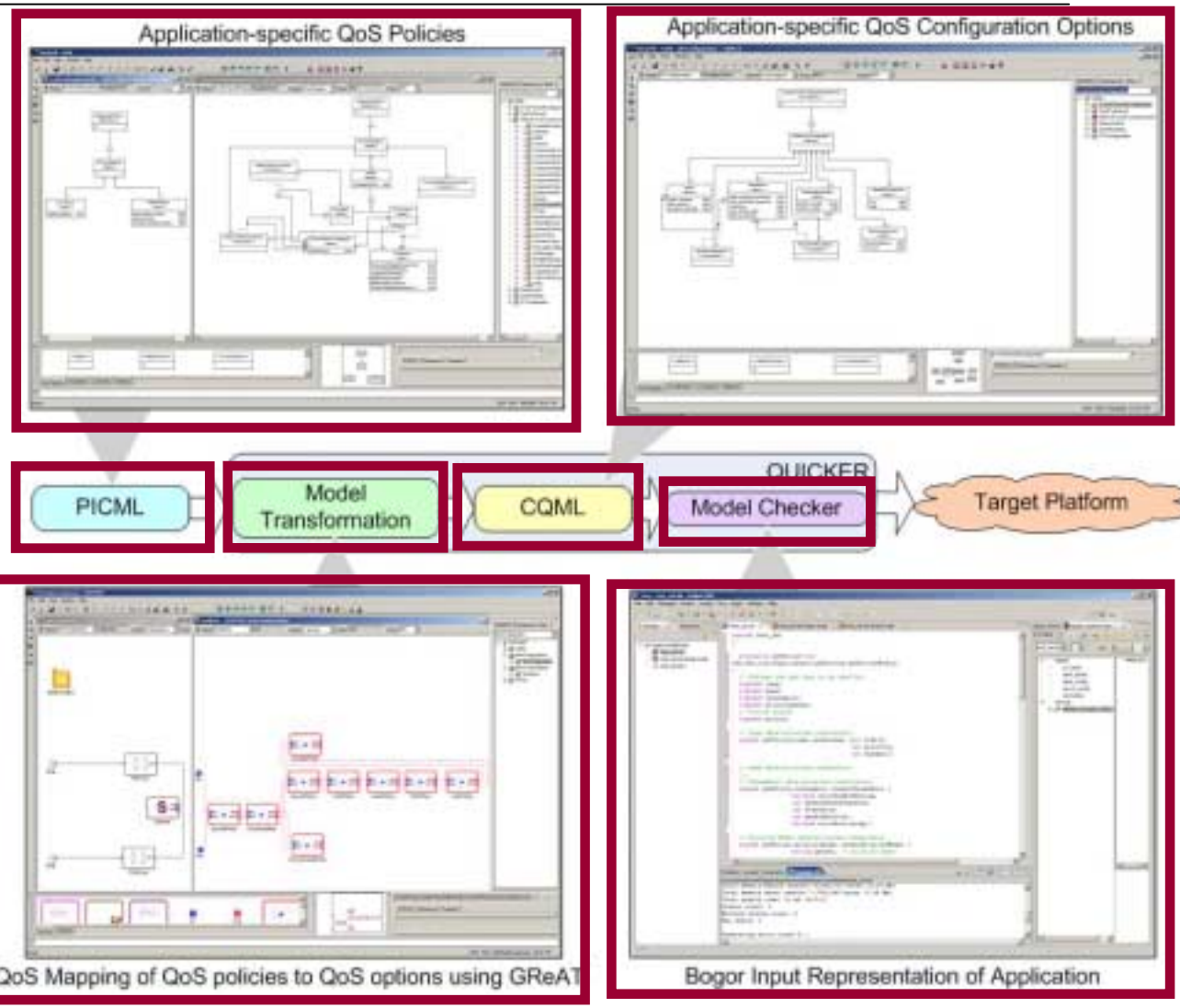
Challenge 4: Resolving QoS Option Dependencies



- "QoS option dependency" is defined as:
  - Dependency between QoS options of different components
- Manually tracking dependencies is hard – or in some cases infeasible
  - Dependent components may belong to more than one assembly
  - Dependency may span beyond immediate neighbors
    - e.g., dependency between Gizmo & Comm components
- Empirically validating configuration changes by hand is tedious, error-prone, & slows down development & QA process considerably
  - Several iterations before desired QoS is achieved (if at all)

Solution Approach: Model-Driven QoS Mapping

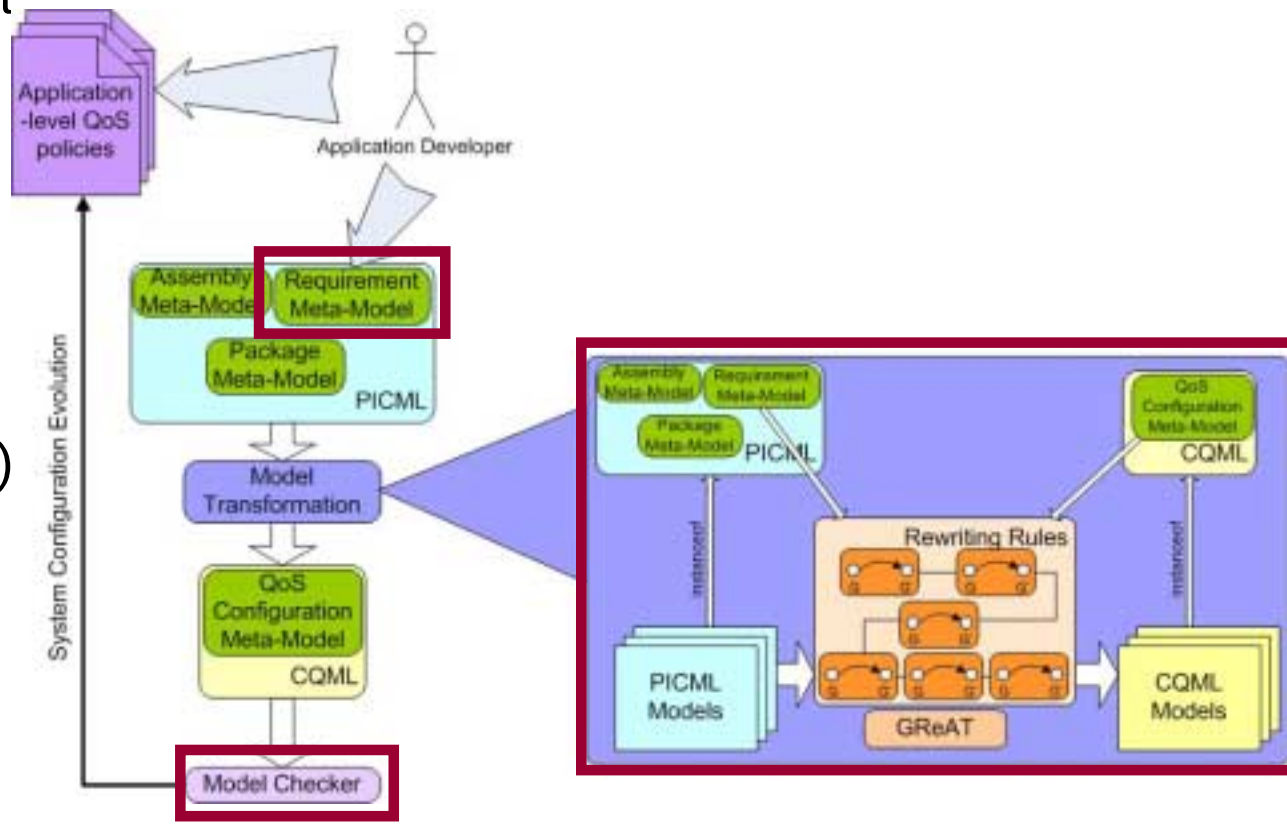
- QUality of service pICKER (QUICKER)
  - Model-driven engineering (MDE) tools model application QoS policies
  - Provides automatic mapping of QoS policies to QoS configuration options
  - Validates the generated QoS options



- Automated QoS mapping & validation tools can be used iteratively throughout the development process

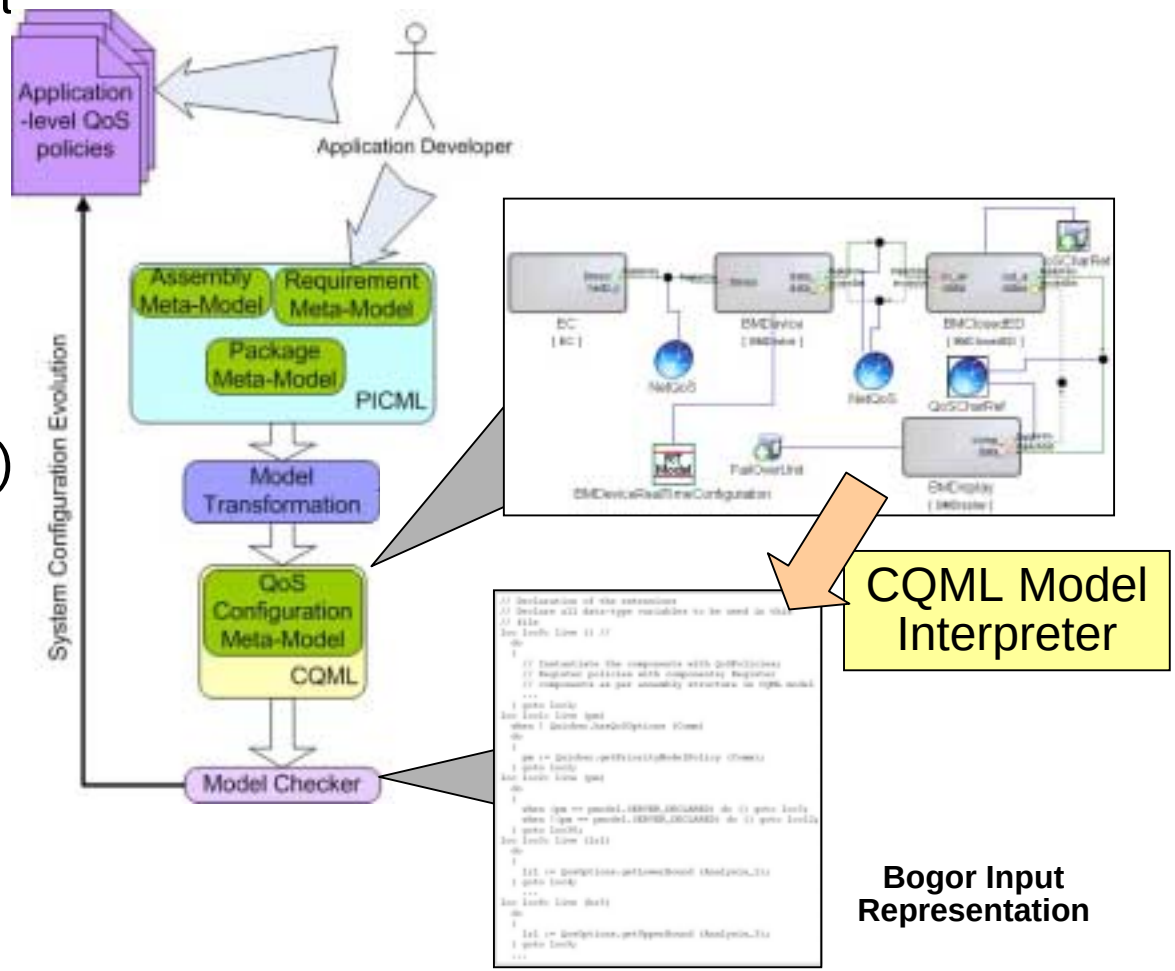
QUICKER Enabling MDE Technologies

- Enhanced Platform Independent Component Modeling Language (PICML), a DSML for modeling component-based applications
- QoS mapping uses Graph Rewriting & Transformation (GReAT) model transformation tool
- Customized Bogor model-checker used to define new types & primitives to validate QoS options



QUICKER Enabling MDE Technologies

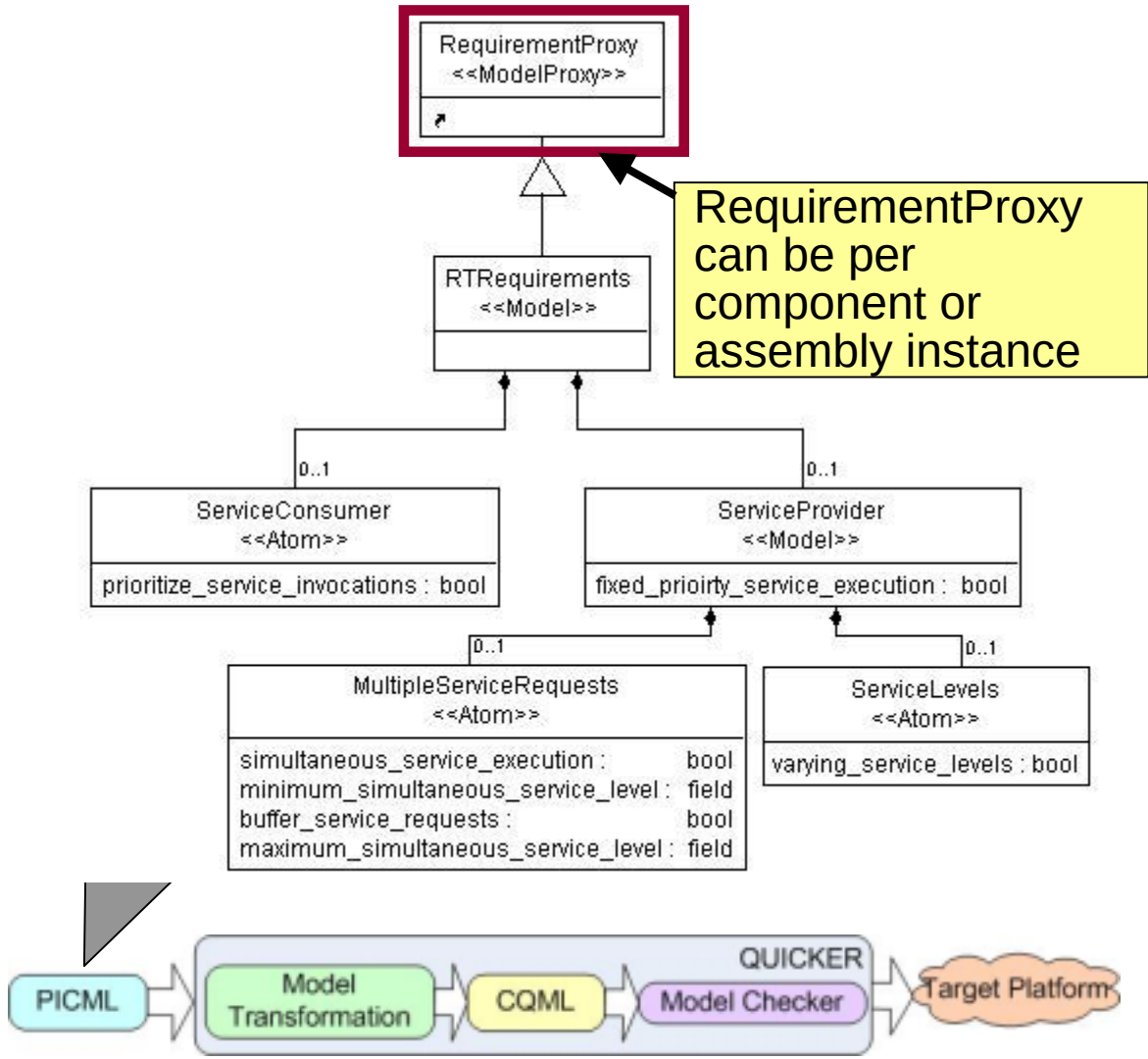
- Enhanced Platform Independent Component Modeling Language (PICML), a DSML for modeling component-based applications
- QoS mapping uses Graph Rewriting & Transformation (GReAT) model transformation tool
- Customized Bogor model-checker used to define new types & primitives to validate QoS options



- CQML Model interpreter generates Bogor Input Representation (BIR) of DRE system from its CQML model

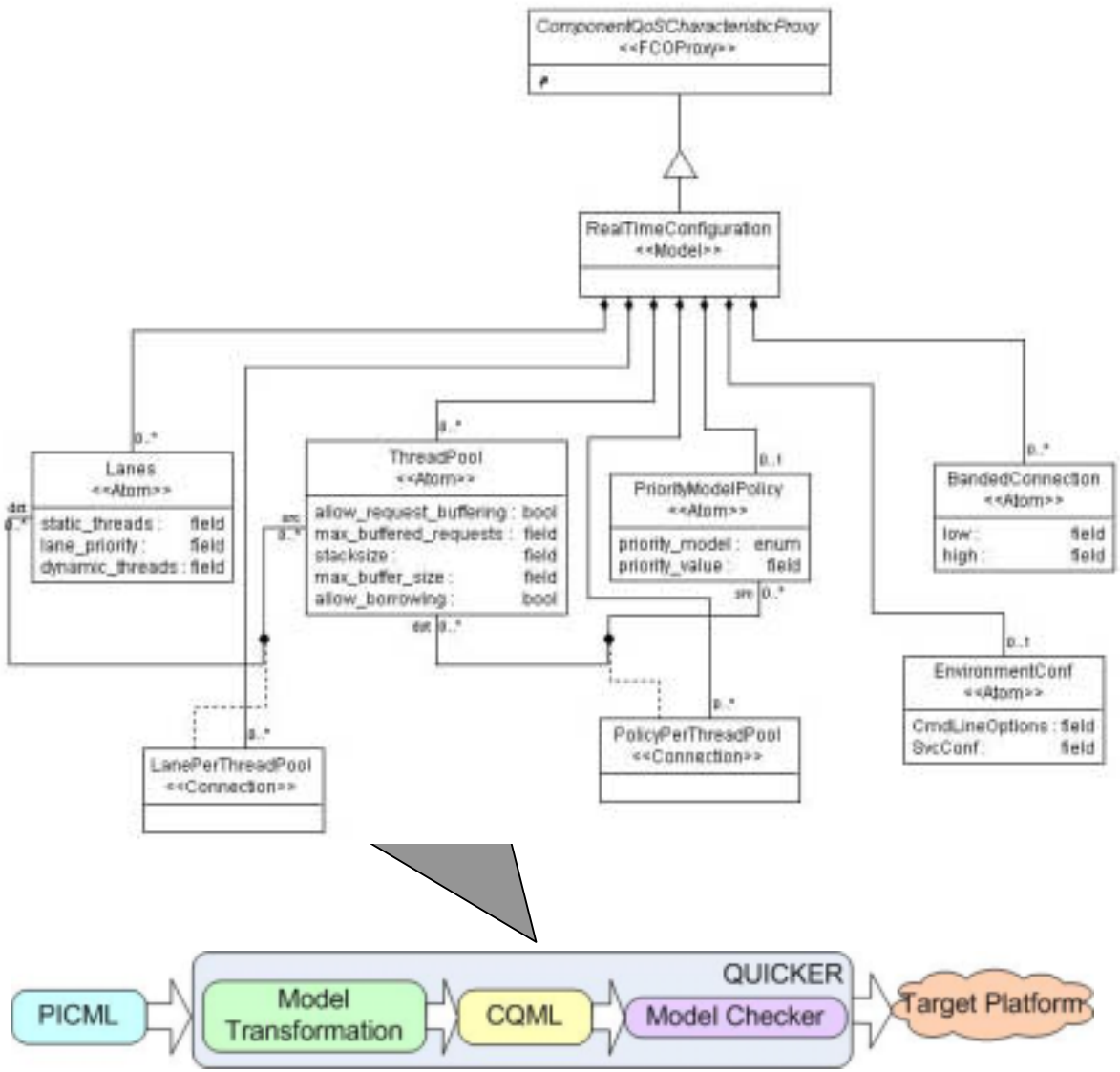
QUICKER Concepts: Transformation of QoS policies(1/2)

1. Platform-Independent Modeling Language (PICML) represents application QoS policies
   - PICML captures policies in a platform-independent manner
   - Representation at multiple levels
     - e.g., component- or assembly-level



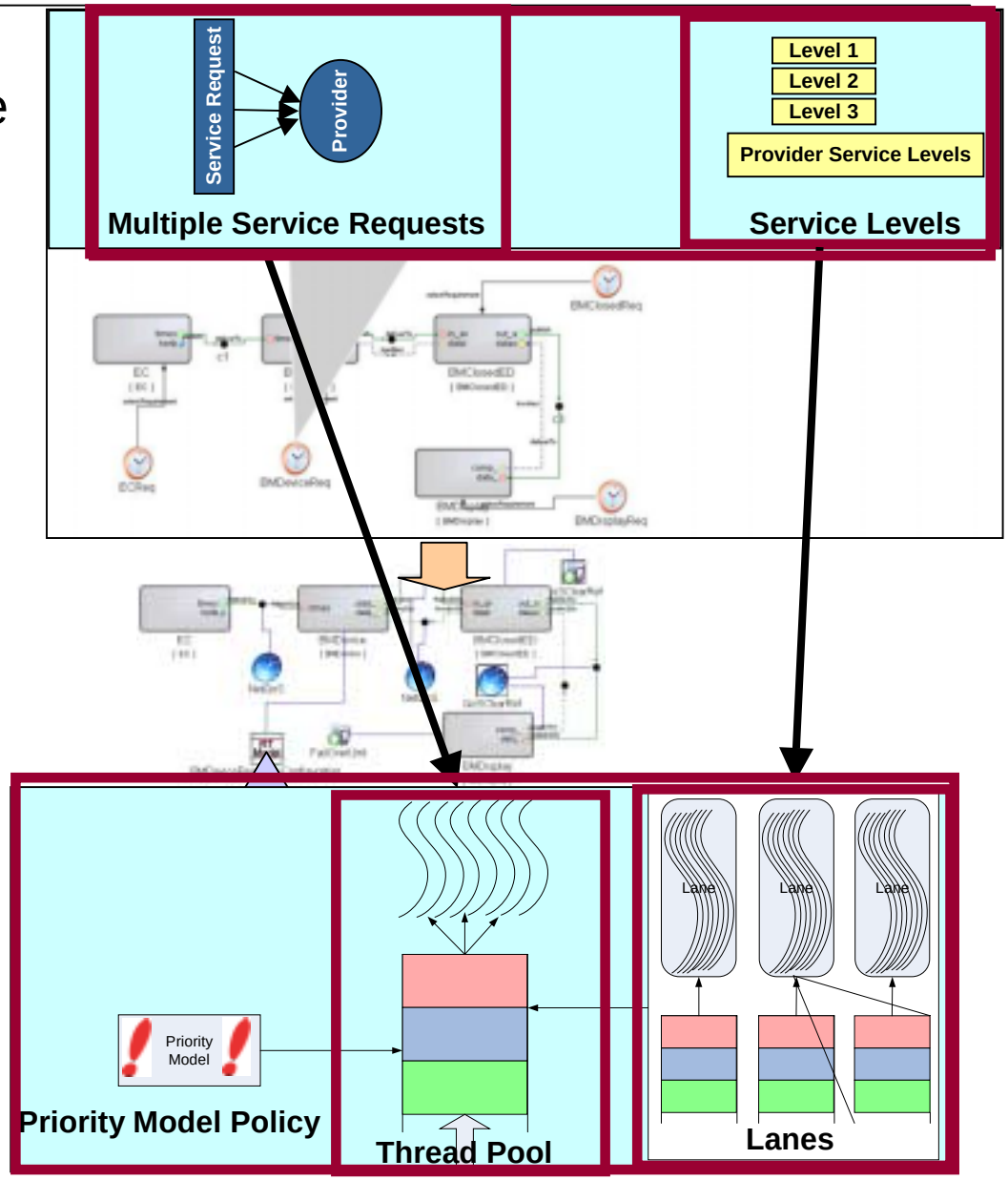
QUICKER Concepts: Transformation of QoS policies(1/2)

1. Platform-Independent Modeling Language (PICML) represents application QoS policies
   - PICML captures policies in a platform-independent manner
   - Representation at multiple levels
     - e.g., component- or assembly-level
2. Component QoS Modeling Language (CQML) represents QoS options
   - CQML captures QoS configuration options in a platform-specific manner



QUICKER Concepts: Transformations of QoS policies(2/2)

3. Translation of application QoS policies into middleware QoS options
   - Semantic translation rules specified in terms of input (PICML) & output (CQML) type graph
     - e.g., rules that translate multiple application service requests & service level policies to corresponding middle-ware QoS options
   - QUICKER transformation engine maps QoS policies (in PICML) to QoS configuration options (in CQML)



QUICKER Concepts: Validation of QoS Options (1/2)

1. Representation of middleware QoS options in Bogor model-checker
   - BIR extensions allow representing domain-level concepts in a system model
   - QUICKER defines new BIR extensions for QoS options
     - Allows representing QoS options & domain entities directly in a Bogor input model
       - e.g., CCM components, Real-time CORBA lanes/bands are first-class Bogor data types
   - Reduces size of system model by avoiding multiple low-level variables to represent domain concepts & QoS options

```
extension QoSOptions for
edu.ksu.cis.bogor.module.QoSOptions.QoSOptionsModule
{
  // Defines the new type to be used for
  typedef lane;
  typedef band;
  typedef threadpool;
  typedef prioritymodel;
  typedef policy;
  expdef QoSOptions.lane createLane (
   int static, int priority, int dynamic);
  // ThreadPool constructor.
  expdef QoSOptions.threadpool
  createThreadPool (boolean allowreqbuffering,
   int maxbufferedrequests, int stacksize, int
   maxbuffersize, boolean allowborrowing);
  // Set the band(s) for QoS policy.
  actiondef registerBands (QoSOptions.policy
   policy, QoSOptions.band ...);
  // Set the lane(s) for QoS policy.
  actiondef registerLanes (QoSOptions.policy
   policy, QoSOptions.lane ...);
  ...
}
extension Quicker for
edu.ksu.cis.bogor.module.Quicker
{
  // Defines the new type.
  typedef Component;
  // Component Constructor.
  expdef Quicker.Component
  createComponent (string component);
  // Set the QoS policy for the component.
  actiondef registerQoSOptions (Quicker.Component
    component,QoSOptions.policy policy);
  // Make connections between components.
  actiondef connectComponents (Quicker.Component
    server,Quicker.Component client);
  ...
}
```

QUICKER Concepts: Validation of QoS Options (2/2)

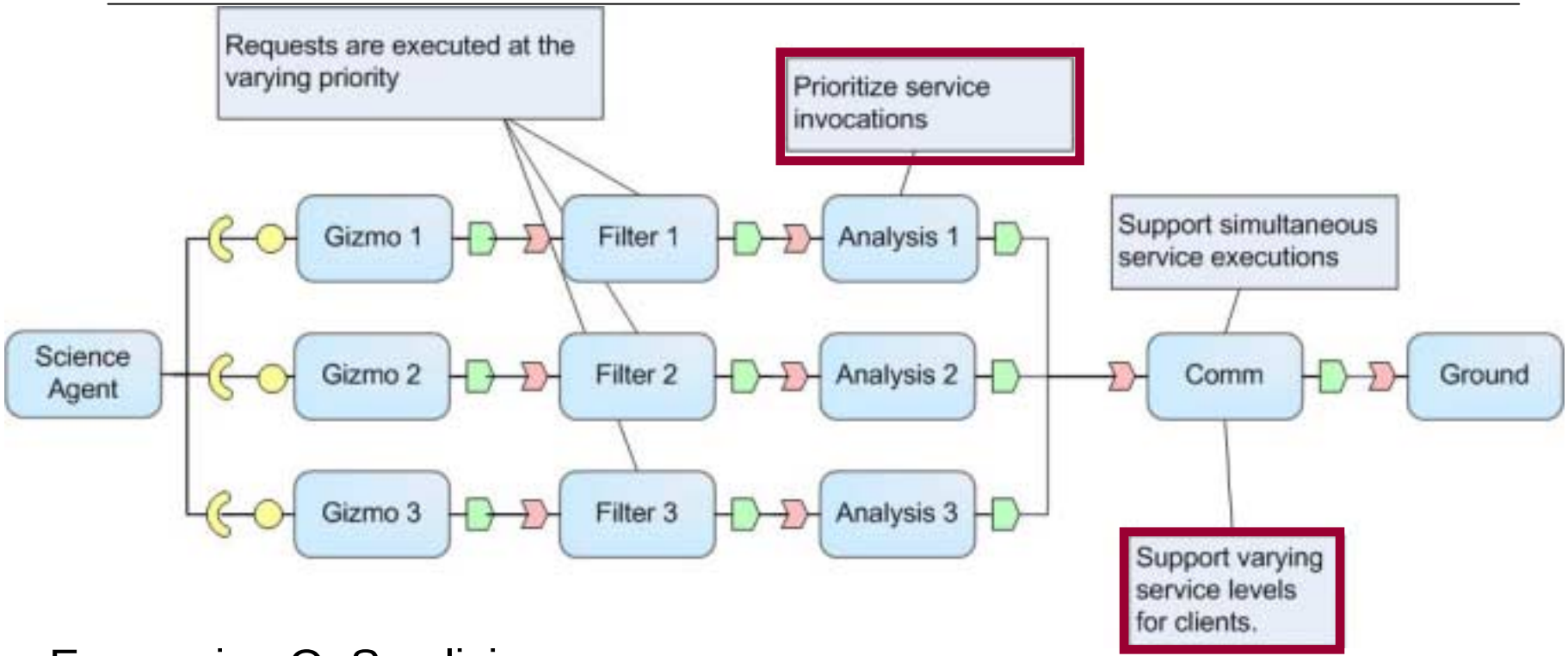
2. Representation of properties (that a system should satisfy) in Bogor
   - BIR primitives define language constructs to access & manipulate domain-level data types, e.g.:
     - Used to define rules that validate QoS options & check if property is satisfied
3. Automatic generation of BIR of DRE system from CQML models

Rule determines if ThreadPool priorities at Comm match priority bands at Analysis

Model interpreters auto-generate Bogor Input Representation of a system from its CQML model

```
// Declaration of the extensions
// Declare all data-type variables to be used in this
// file
loc loc0: live () //
  do
  {
    // Instantiate the components with QoSPolicies;
    // Register policies with components; Register
    // components as per assembly structure in CQML model
    ...
  } goto loc1;
loc loc1: live (pm)
  when ! Quicker.hasQoSOptions (Comm)
  do
  {
    pm := Quicker.getPriorityModelPolicy (Comm);
  } goto loc2;
loc loc2: live (pm)
  do
  {
    when (pm == pmodel.SERVER_DECLARED) do {} goto loc3;
    when !(pm == pmodel.SERVER_DECLARED) do {} goto loc12
  } goto loc30;
loc loc3: live (lr1)
  do
  {
    lr1 := QosOptions.getLowerBound (Analysis_1);
  } goto loc4;
loc loc8: live (hr3)
  do
  {
    lr1 := QosOptions.getUpperBound (Analysis_3);
  } goto loc9;
  ...
```

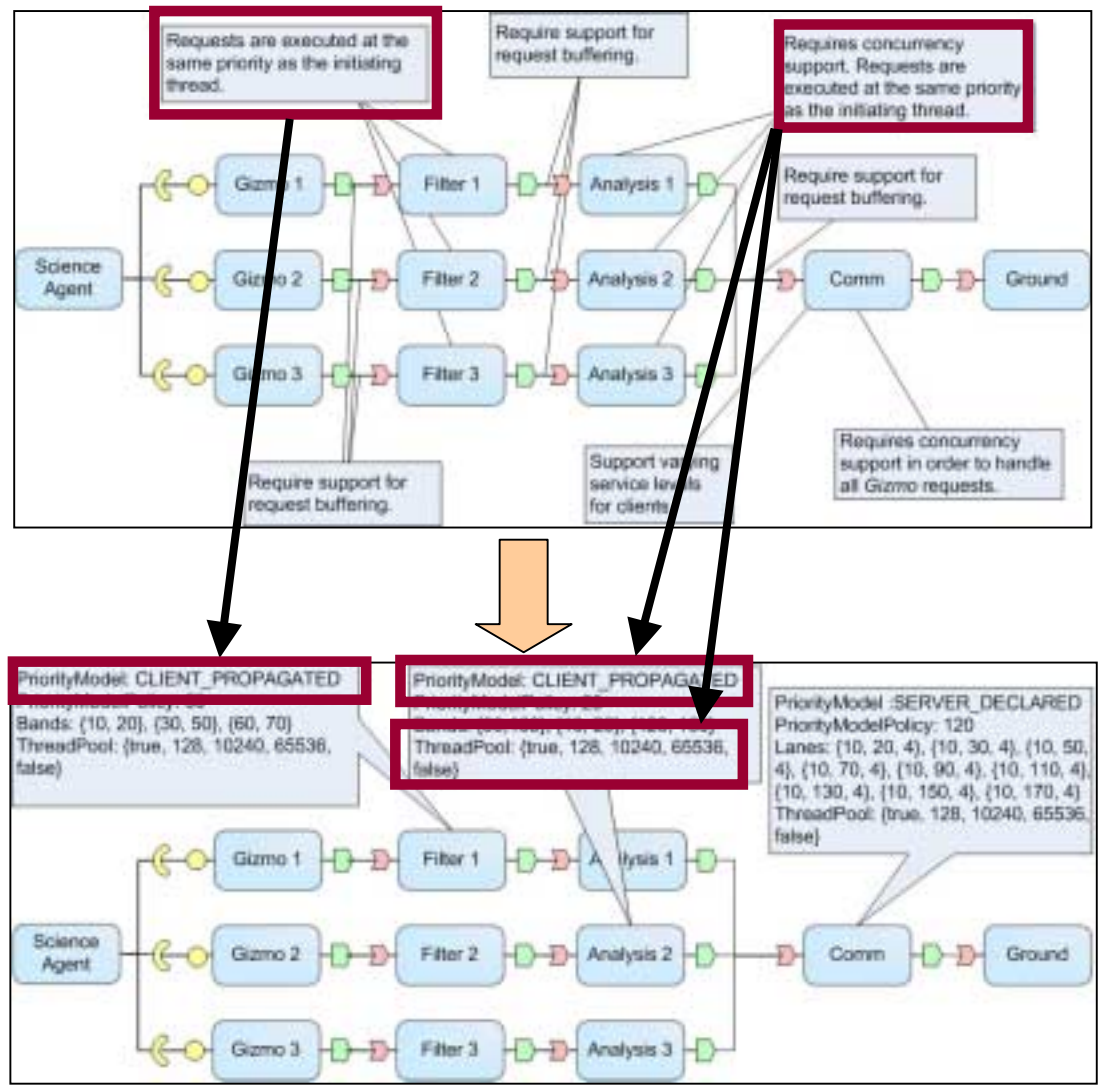
Resolving Challenge 1: Translating Policies to Options (1/2)



- Expressing QoS policies
  - PICML modes application-level QoS policies at high-level of abstraction
    - e.g., multiple service levels support for Comm component, service execution at varying priority for Analysis component
  - Reduces modeling effort
    - e.g., ~25 QoS policy elements for MMS mission vs. ~140 QoS options

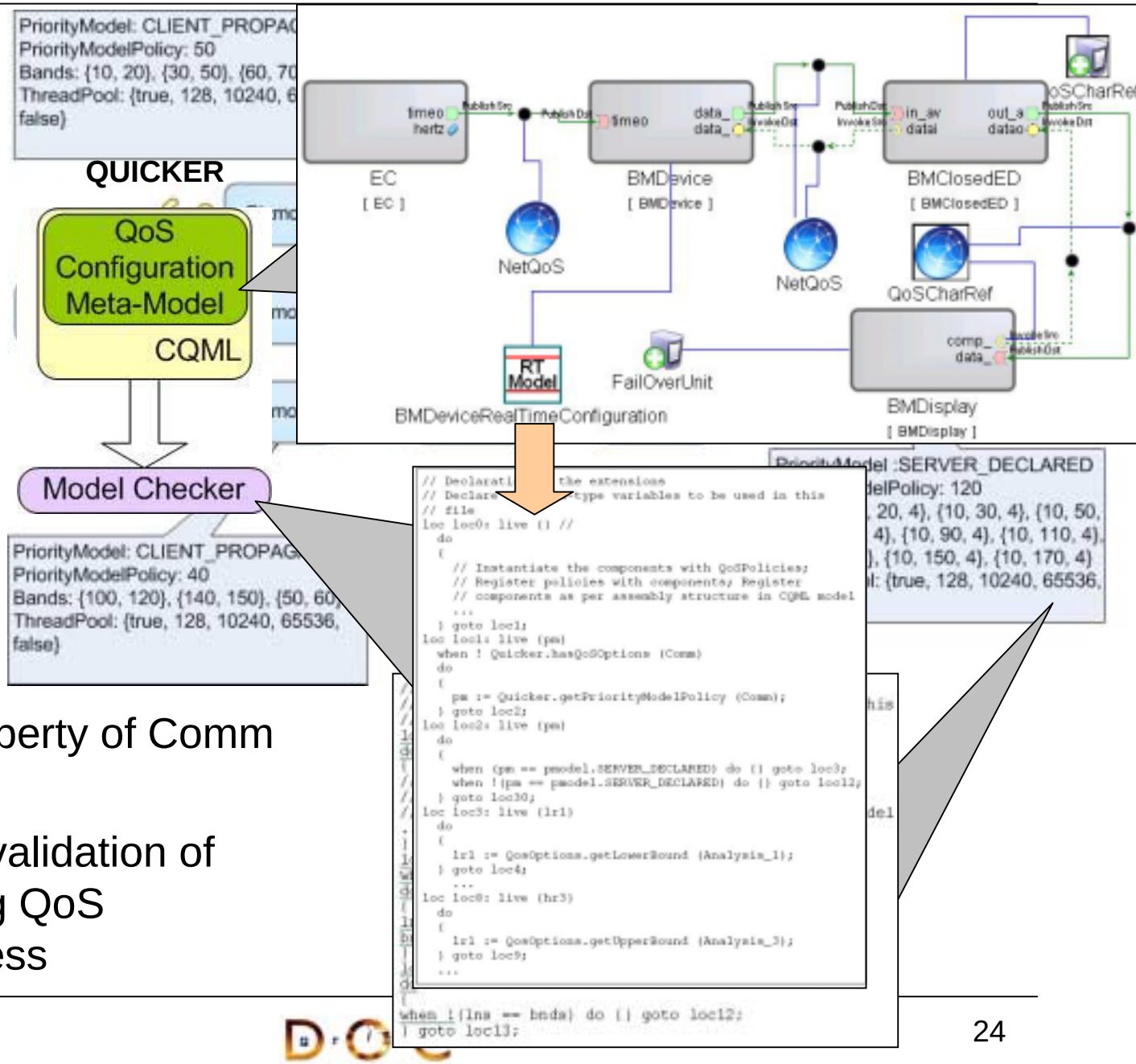
Resolving Challenge 1: Translating Policies to Options (2/2)

- Mapping QoS policies to QoS options
  - GReAT model transformations automate the tedious & error-prone translation process
- Transformations generate QoS configuration options as CQML models
  - Allow further transformation by other tools
    - e.g., code optimizers & generators
  - Simplifies application development & enhances traceability



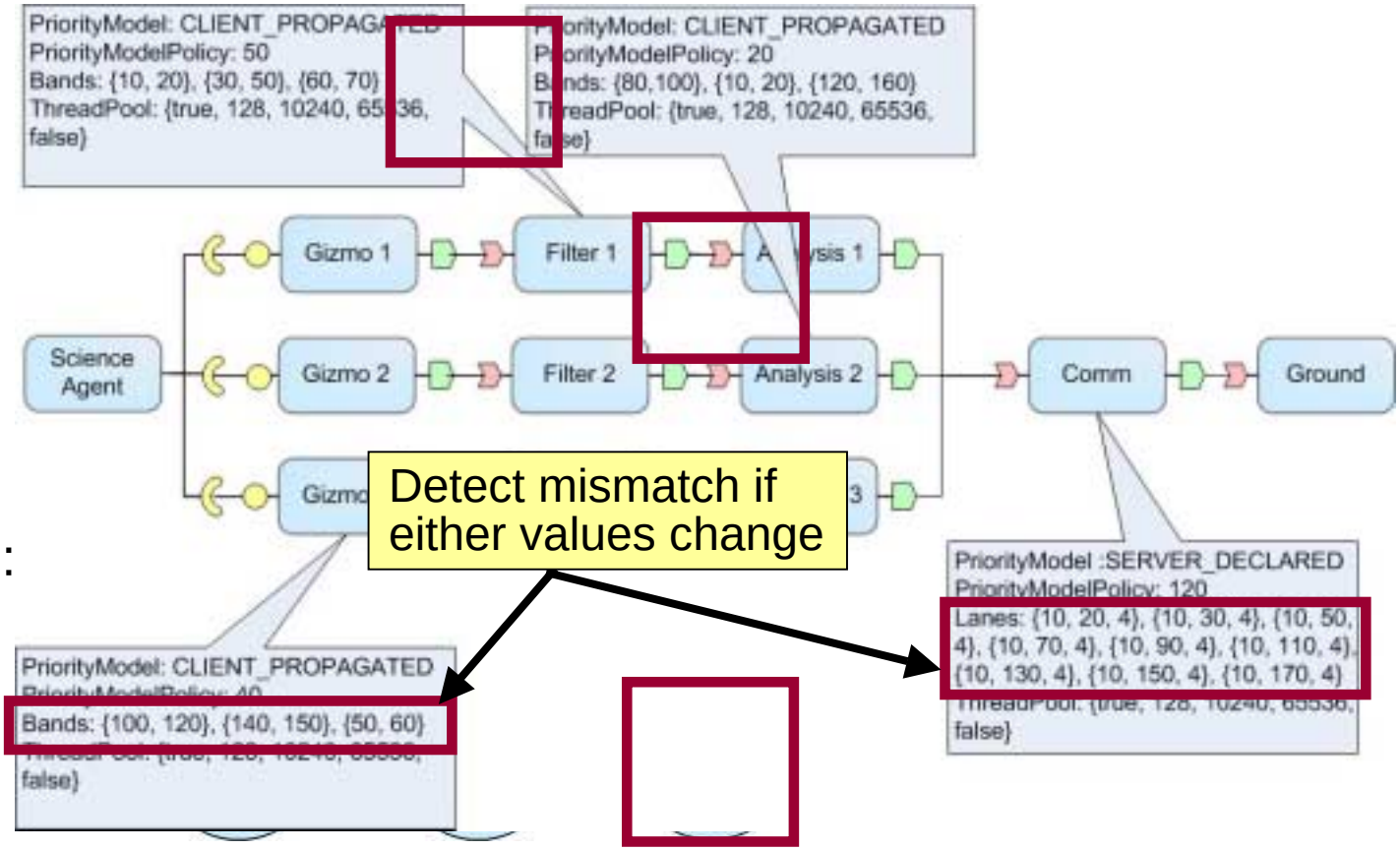
Resolving Challenges 2 & 3: Ensuring QoS Option Validity

- CQML model interpreter generates BIR specification from CQML models
- BIR primitives used to check whether a given set of QoS options satisfies a system property
  - e.g., fixed priority service execution, a property of Comm component
- Supports iterative validation of QoS options during QoS configuration process



Resolving Challenge 4: Resolving QoS Option Dependencies

- Dependency structure maintained in Bogor used to track dependencies between QoS options of components, e.g.:
  - Analysis & Comm are connected
  - Gizmo & Comm are dependent



Dependency Structure of MMS Mission Components

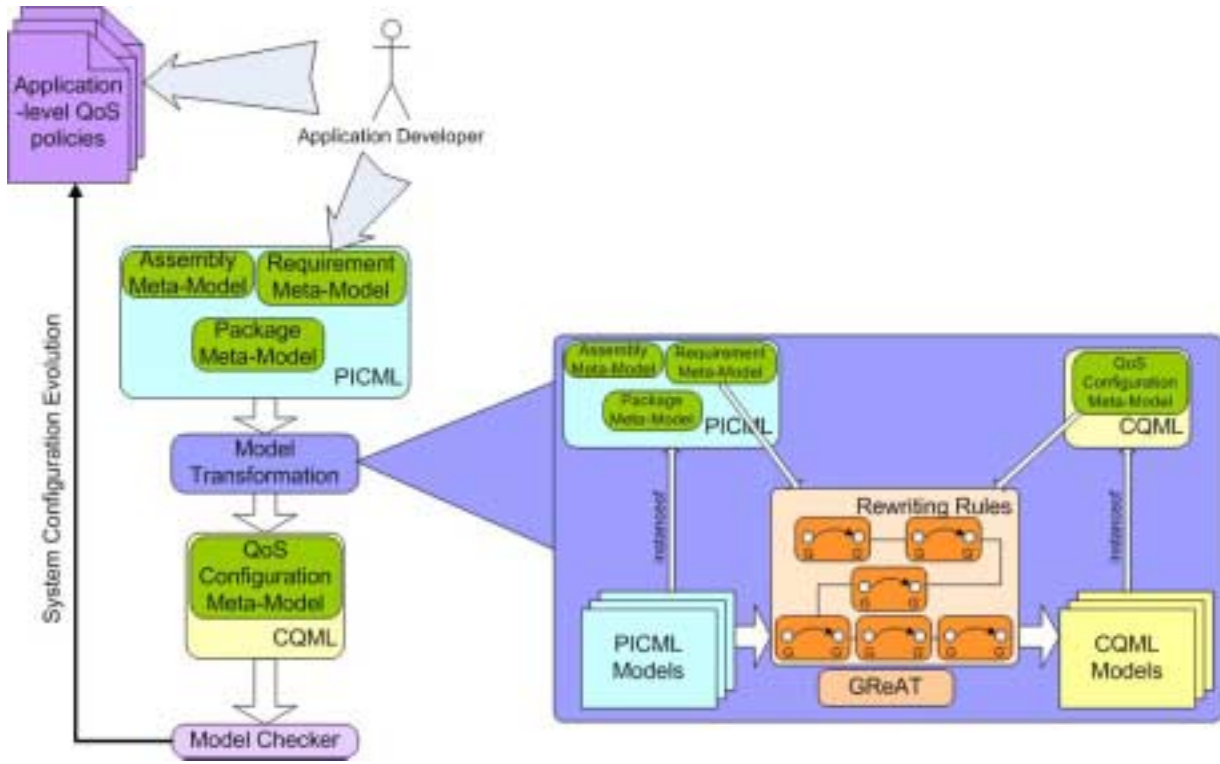
- Change(s) in QoS options of dependent component(s) triggers detection of potential mismatches
  - e.g., dependency between Gizmo invocation priority & Comm lane priority

Academic Related Work

- Functional Specification & Analysis Tools
  - Hatcliff, J. et. al. (2003). *Cadena*
- QoS Adaptation Modeling Tools
  - Ye, J. et. al. (2004). *DQME*
  - Zinky, J., (1997). QuO
- QoS Specification Tools
  - Ritter, T. et. al. (2003). *CCM QoS MetaModel*
  - Ahluwalia, J. et. al. (2005). *Model-based Run-tim Monitoring*
  - Frolund, S. et. al. (1998). *QML*
- Schedulability Analysis Tools
  - Madl, G. et. al. (2004). *Automatic Component-based system verification*
  - Kodase, S. et. al. (2003). *AIRES*

Concluding Remarks

- QUICKER provides Model-Driven Engineering (MDE) for QoS-enabled component middleware
- Maps application-level QoS policies to middleware-specific QoS configuration options
- Model transformations automatically generate QoS options
- Model-checking extensions ensure validity of QoS options at component- & application-level



QUICKER MDE tools & CIAO QoS-enabled component middleware available as open-source at www.dre.vanderbilt.edu/CoSMIC

Questions?