Empirical Evaluation of CORBA Component Model Implementations

Arvind S. Krishna,
Douglas C. Schmidt et.al
Institute for Software
Integrated Systems (ISIS)
{arvindk, schmidt}@dre.vanderbilt.edu

Gautam Thaker
Lockheed Martin Advanced
Technology Labs
gthaker@atl.lmco.com

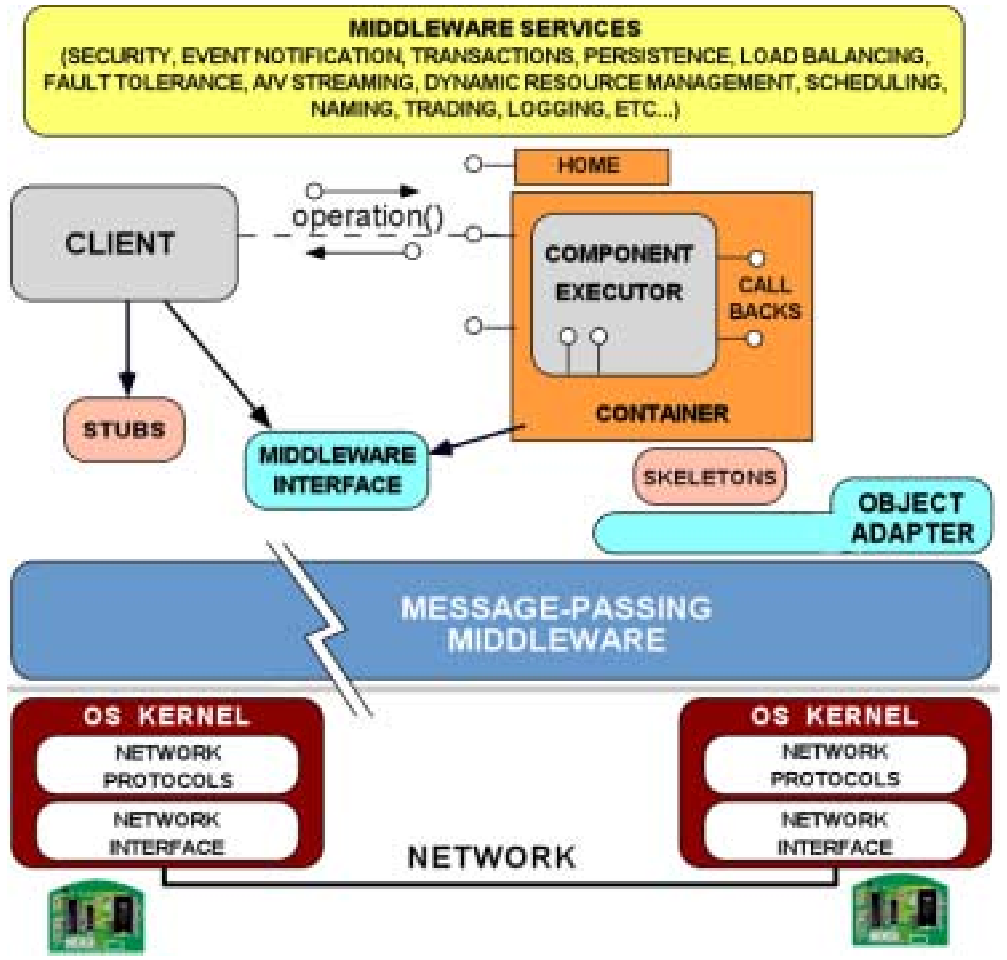
Nanbor Wang
Tech X. Corporation
nanbor@cs.wustl.edu

Diego Sevilla Ruiz
University of Murcia, Spain
diego@ditec.um.es

Talk Outline

- Technology Trends – Motivation for CCMPerf
- CCMPerf Benchmark Design
    - Experimentation Categories
    - Metrics Gathered
    - Experimentation Results (Distribution overhead results)
- Generative Focus for Benchmark Synthesis
    - Benchmark Generation Modeling Language
    - Modeling Avionics Scenario using BGML
- Concluding Remarks & Future Work

Technology Trends – Commoditization



Information technology is being commoditized

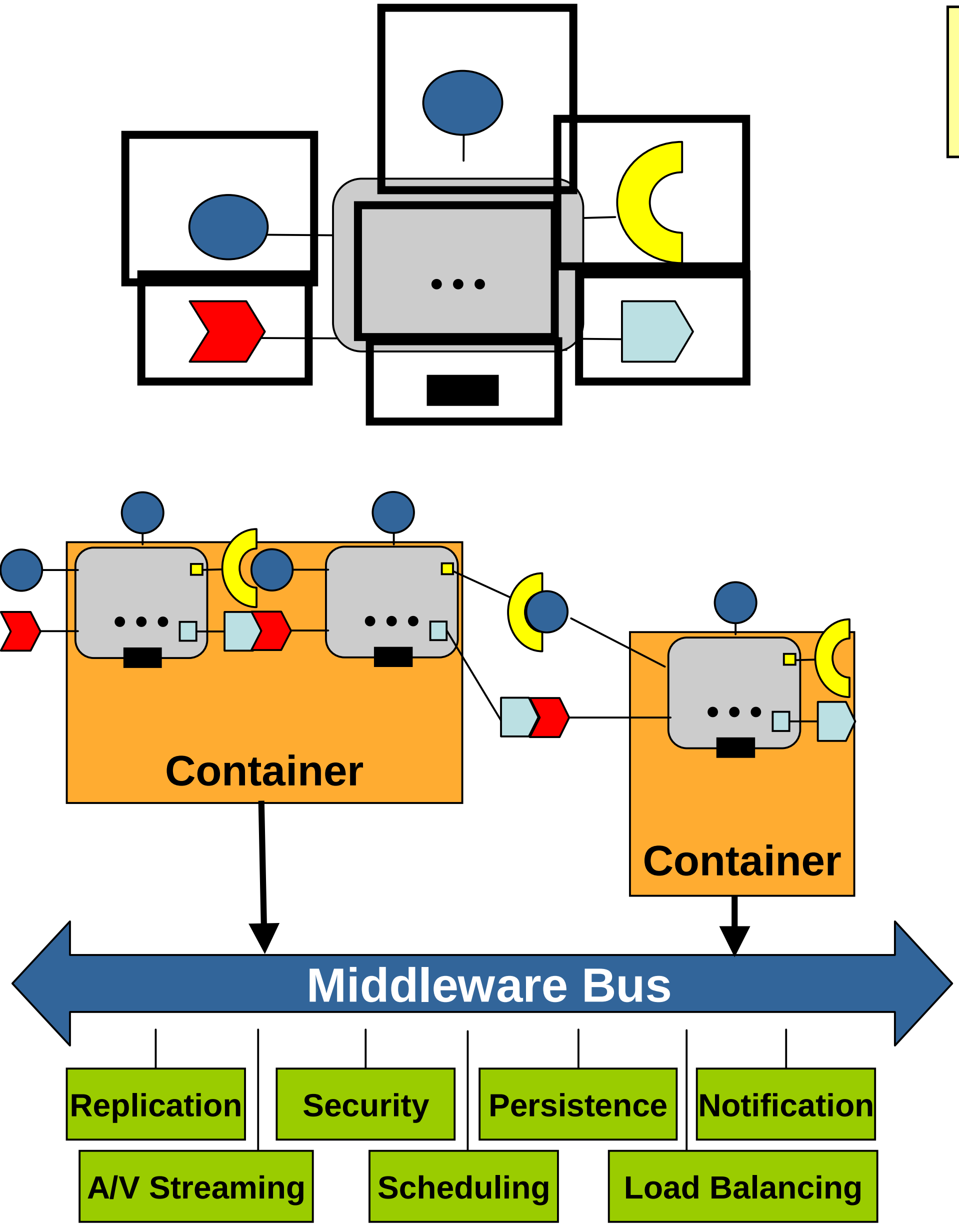
- i.e., hardware & software are getting cheaper, faster, & (generally) better at a fairly predictable rate

Growing acceptance of a network-centric component paradigm

i.e., distributed applications with a range of QoS needs are constructed by integrating components & frameworks via various communication mechanisms

These advances stem largely from standard hardware & software APIs, network protocols, and middleware

Technology Trends – Component Middleware



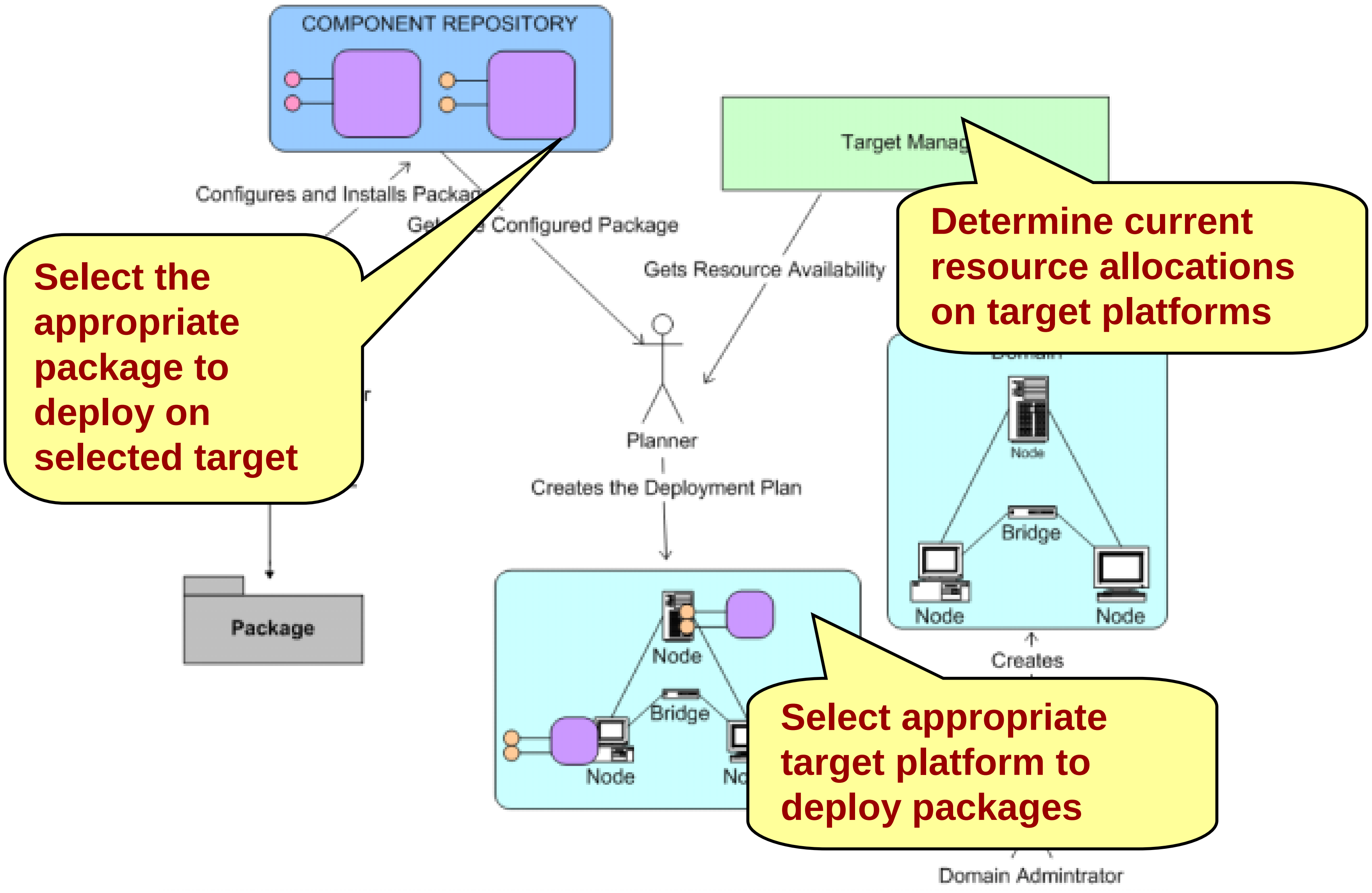
Component middleware is maturing & becoming pervasive

- *Components* **encapsulate application "business" logic**
- **Components interact via** *ports*
  - ***Provided interfaces***, e.g., facets
  - ***Required connection points***, e.g., receptacles
  - ***Event sinks & sources***
  - ***Attributes***
- *Containers* **provide execution environment for components with common operating requirements**
- **Components/containers can also**
  - Communicate via a ***middleware bus*** and
  - Reuse ***common middleware services***

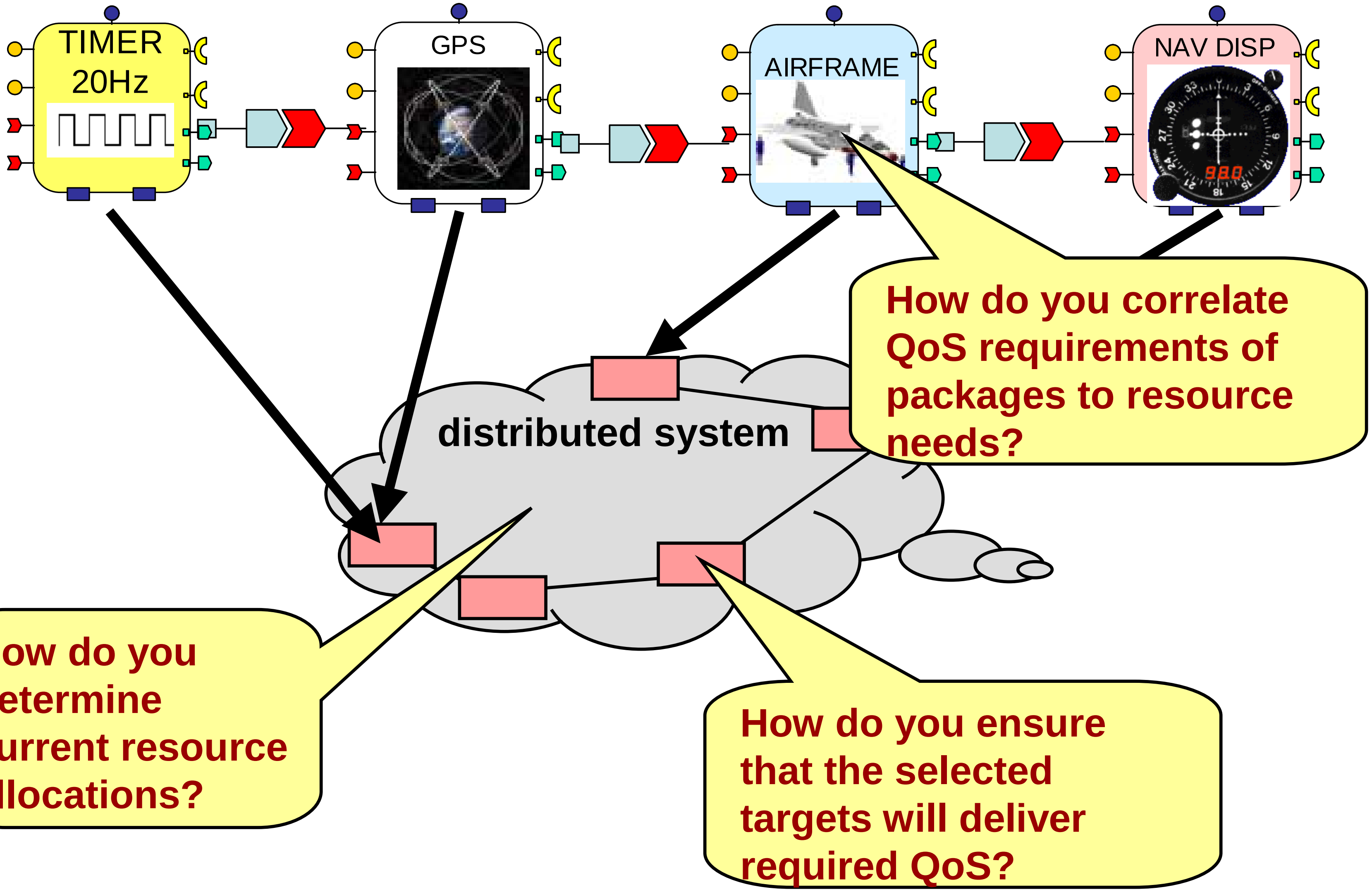
Technology Trends – DnC Specification

- ***Packaging***
  - bundling a suite of software binary modules and metadata representing application components
- ***Installation***
  - populating a repository with the packages required by the application
- ***Configuration***
  - configuring the packages with the appropriate parameters to satisfy the functional and systemic requirements of application without constraining to any physical resources
- ***Planning***
  - making appropriate deployment decisions including identifying the entities, such as CPUs, of the target environment where the packages will be deployed
- ***Preparation***
  - moving the binaries to the identified entities of the target environment
- ***Launching***
  - triggering the installed binaries and bringing the application to a ready state
- ***Adaptation***
  - Runtime reconfiguration & resource management to maintain end-to-end QoS

Planning Aspect – Challenges (1/2)


COMPONENT REPOSITORY
Configures and Installs Packag
Ge Configured Package
Target Manag
Gets Resource Availability
Planner
Creates the Deployment Plan
Package
Node
Bridge
Node
Node
Node
Bridge
Node
Creates
Domain Admintrator
Select the appropriate package to deploy on selected target
Determine current resource allocations on target platforms
Select appropriate target platform to deploy packages

Planning Aspect – Challenges (2/2)



How do you correlate QoS requirements of packages to resource needs?

How do you determine current resource allocations?

How do you ensure that the selected targets will deliver required QoS?

Technology Trends – Motivation for CCMPerf

Component middleware is increasingly used to build large scale applications, CCM Implementations also increase

Open-Source/GPL versions:

- CIAO – Washington Univ. St Louis & Vanderbilt Univ.
- MICO-CCM – MICO ORB CCM Implementation
- Start CCM – China

Commercial versions:

- K2 – ICMG Corporation, Bangalore, India

However, no body of knowledge to analyze the following:

- Suitability → how suitable is the CCM implementation for DRE applications in a particular domain, such as avionics, total ship computing, or telecom?
- Quality → How good is the CCM implementation? For example, in the DRE domain, is the implementation predictable
- Correctness → Does the implementation correspond to the OMG specification portability and interoperability requirements?

Talk Outline

- Technology Trends – Motivation for CCMPerf
- CCMPerf Benchmark Design
  - Experimentation Categories
  - Metrics Gathered
  - Experimentation Results (Distribution overhead results)
- Generative Focus for Benchmark Synthesis
  - Benchmark Generation Modeling Language
  - Modeling Avionics Scenario using BGML
- Concluding Remarks & Future Work

Design of CCMPerf

Challenges

• Heterogeneity in implementations → header files, descriptor files different

•Difference in the domain of application → use cases change e.g. CIAO tailored towards DRE applications

•Heterogeneity in software configuration options → Different implementations provide different knobs to fine tune performance

Benchmark experimentation categories

•Distribution Middleware Benchmarks → quantify the overhead of CCM-based applications relative to normal CORBA

• Common Middleware Services Benchmarks → quantify suitability of different implementations of common CORBA services

• Domain Specific Benchmarks → tests that quantify the suitability of CCM implementations to meet the QoS requirements of a particular DRE application domain

Developing Metrics (1/2)

Metrics Gathered in each Benchmark category

Every benchmarking category of experiments has white box and black box related metrics

Black-box related Metrics

- Round-trip latency measures, response time for a two-way operation
- Throughput, the number of events per second at client
- Jitter, the variance in round-trip latency
- Collocation performance, which measures response time and throughput when a client and server are in the same process
- Data copying overhead, which compares the variation in response time with an increase in request size CCMPerf measures each of these metrics in single-threaded and multi-threaded configurations on both servers and clients.

Developing Metrics (2/2)

White-box related metrics

- Functional path analysis identify CCM layers that are above the ORB and adds instrumentation points to determine the time spent in those layers.
- Lookup time analysis measure the variation in lookup time for certain operations, such as finding component homes, obtaining facets, etc
- Context switch times measure the time required to interrupt the currently running thread and switch to another thread in multi-threaded implementations.

Experimentation	White-box metrics	Black-box metrics
Distribution Middleware	Functional path & Jitter	Throughput, latency foot-print & collocation
Common Middleware services	Jitter Analysis Context switch times	Throughput, latency & collocation
Domain specific benchmarks	QoS specific measures	Latency, throughput & Jitter

Distribution Middleware Benchmarks (1/4)

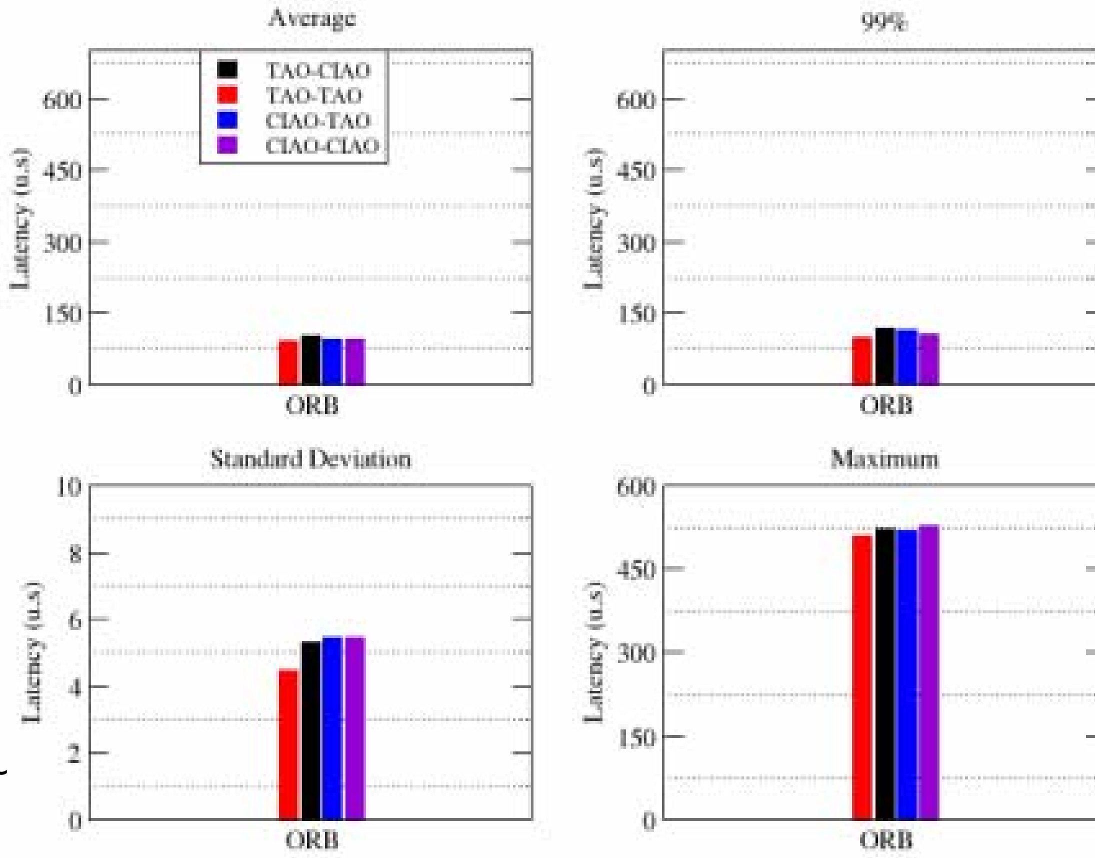
Overview

Four cases in which Component middleware can be mixed and matched with DOC middleware. Measure simple two way round-trip latency

- TAO-CIAO (A CORBA 2.x server with a CCM Client)
- TAO-TAO (CORBA 2.x client and Server)
- CIAO-TAO (CCM Server with a CORBA 2.x Client)
- CIAO-CIAO (CCM Client and Server)

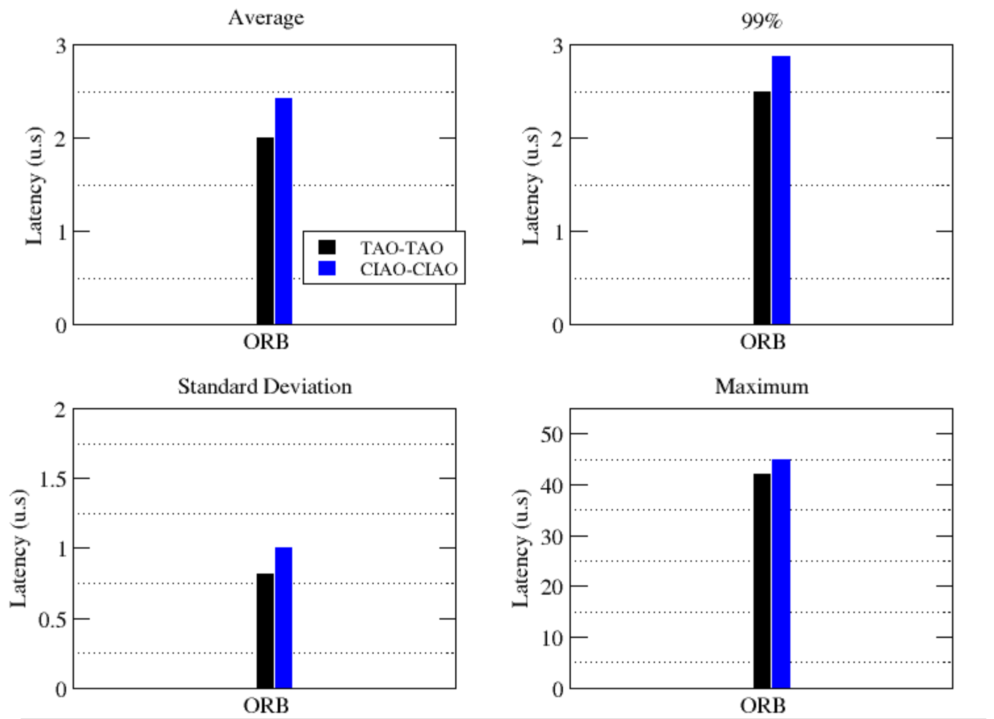
Result Synopsis

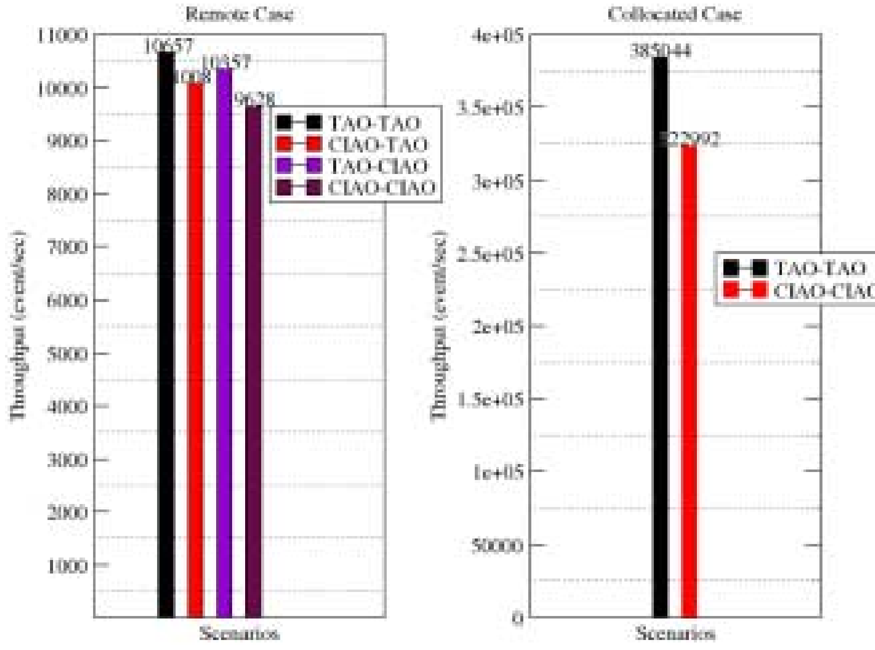
- Average Measures:
  - Comparable results for all four cases (Overhead added by CCM ~ 8 µs)
- Dispersion Measures:
  - Dispersion measures in the same rage (DOC middleware ~ 4.5 vs. CCM 5.2)
- Worst-Case Measures:
  - Use of CCM does not sacrifice predictability



- Use of Component Middleware does not unduly degrade performance of applications
- Average overhead added by CIAO is ~ 5.3 %

Distribution Middleware Benchmarks (2/4)





Collocated Case – Both client and server in the same Address Space

- Compare TAO-TAO vs. CIAO-CIAO case
- Avg latency overhead imparted by CIAO ~ 0.8 µs
- Dispersion measures indicate that CIAO does not degrade predictability by increasing jitter. Same for Worst-case measures

Throughput measured at client at event/sec

- Remote Case
    - TAO-TAO vs. CIAO-CIAO case, overhead added by CIAO ~ 5.6%
- Collocated Case
    - TAO-TAO vs. CIAO-CIAO case overhead added by CIAO ~ 12 %

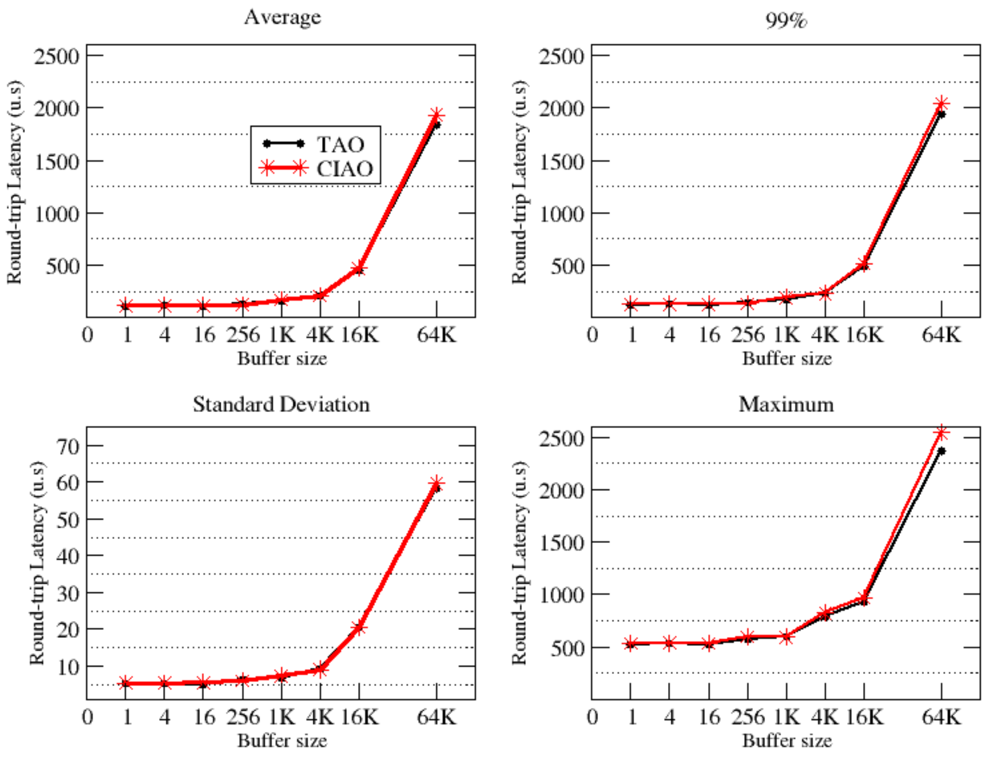
Distribution Middleware Benchmarks (3/4)

Overview

- Measure data-copying overhead for both TAO CIAO
- By exponentially increasing message size check if CIAO incurs any additional data copying overhead than TAO

Result Synopsis

- Average Measures:
    - CIAO does not incur any additional overhead for all cases
- Dispersion Measures:
    - Though dispersion increases with increase in buffer size, CIAO does not add any additional overhead
- Worst-Case Measures:
    - Show a trend similar to that of TAO



Black box metrics show that use of CIAO does not unduly degrade performance or predictability for applications using Component middleware

Distribution Middleware Benchmarks (4/4)

Overview

- Measure the time spent in the Component Middleware layer during a normal request processing
- Measured via using internal timers in the ORB Core

Result Synopsis

– Average Measures:

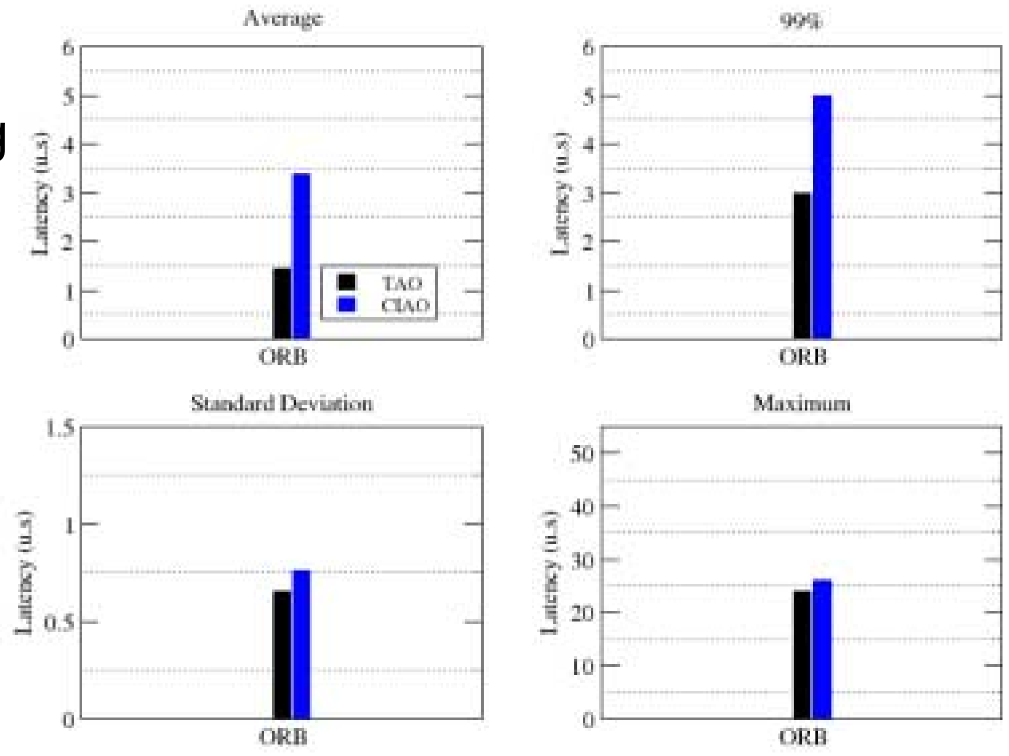
- Latency for normal servant upcall CIAO adds ~ 1.5 µs overhead

–Dispersion Measures:

- Comparable dispersion indicates the overhead does not unduly sacrifice predictability

–Worst-Case Measures:

- Comparable worst-case measures



White-box latency metrics indicate that time spent in the Component layer is ~ 1.5 µs over a normal CORBA upcall processing time. This overhead is incurred by an additional indirection needed to go from the servant to the executor implementation.

Talk Outline

- Technology Trends – Motivation for CCMPerf
- CCMPerf Benchmark Design
  - Experimentation Categories
  - Metrics Gathered
  - Experimentation Results (Distribution overhead results)
- Generative Focus for Benchmark Synthesis
  - Benchmark Generation Modeling Language
  - Modeling Avionics Scenario using BGML
- Concluding Remarks & Future Work

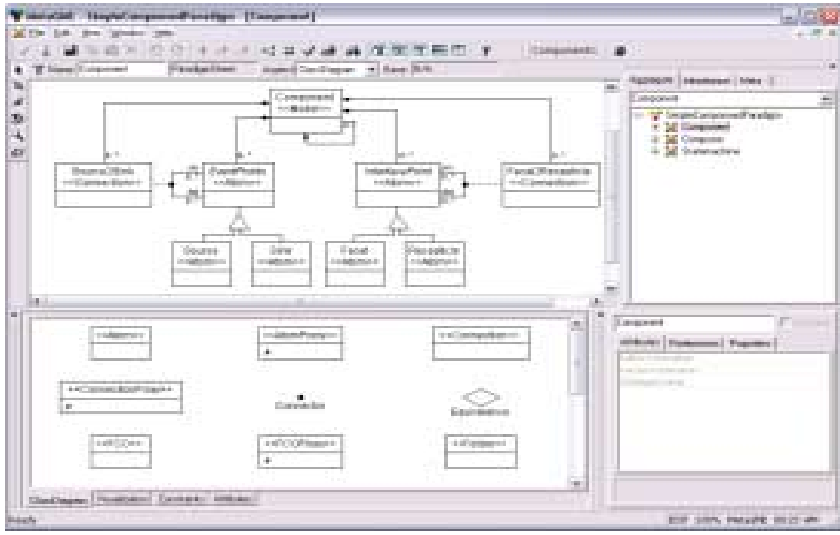
Generative Software Techniques

- Lessons Learned
  - Writing benchmark/test cases is tedious and error-prone
    - For each test one needs to write IDL files, Component IDL files, Executor IDL files, Executor implementation files, Script files for experiment set-up teardown and finally Makefiles!!
  - These have to be carried out for each and every experiment for every scenario, reduces productivity
  - Source code not the right abstraction to visualize these experiments
- Generative Programming Techniques
  - Need to develop tools to auto-generate these benchmarks from higher level models
  - Provide a Domain-Specific (CCM) modeling framework to visually model an interaction scenario
  - Interpreters, then synthesize required benchmarking code from the models

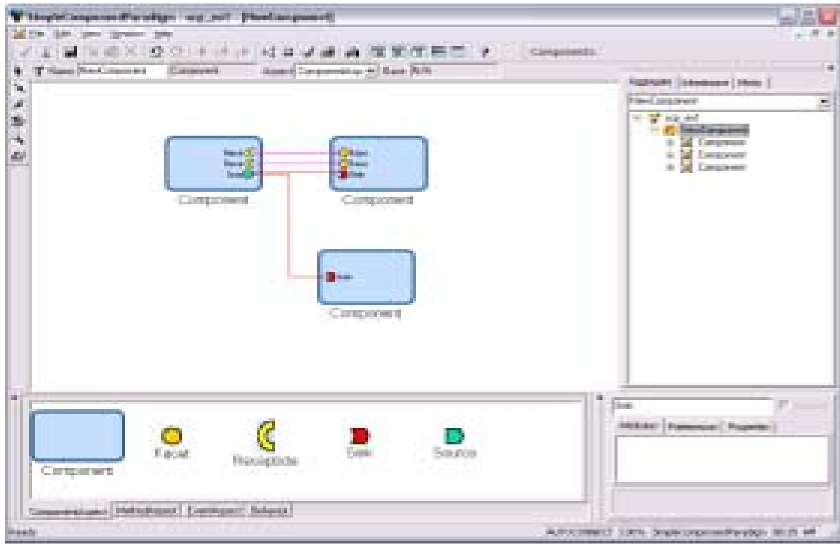
Generic Modeling Environment (GME)

www.isis.vanderbilt.edu/Projects/gme/default.htm

- **Tool Developer (Metamodeler)**
  - –GME used to develop a *domain-specific graphical modeling environment*
  - –Define syntax, static semantics, & visualization of the environment
  - –Semantics implemented via *interpreters*



- **Application Developer (Modeler)**
  - –Uses a specific modeling environment (created by metamodeler w/GME) to build *applications*
  - –The interpreter produces something useful from the models
    - e.g., code, simulations, configurations



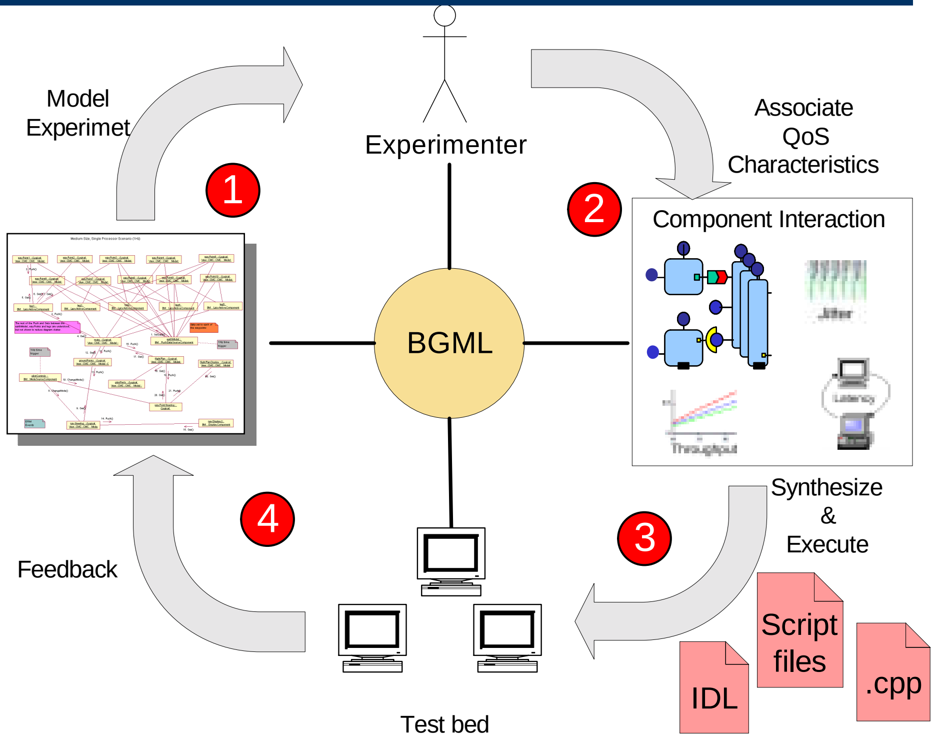
Benchmark Generation Modeling Language

BGML Motivation

- Provide a model-driven tool-suite to empirically evaluate and refine configurations to maximize application QoS

BGML Workflow

1. End-user composes the scenario in the BGML modeling paradigm
2. Associate QoS properties with this scenario, such as latency, throughput or jitter
3. Synthesize the appropriate test code to run the experiment and measure the QoS
4. Feed-back metrics into models to verify if system meets appropriate QoS at design time



- The tool enables synthesis of all the scaffolding code required to set up, run, and tear-down the experiment.
- Using BGML tool it is possible to synthesize :
  - Benchmarking code
  - Component Implementation code
  - IDL and Component IDL files

Talk Outline

- Technology Trends – Motivation for CCMPerf
- CCMPerf Benchmark Design
  - Experimentation Categories
  - Metrics Gathered
  - Experimentation Results (Distribution overhead results)
- Generative Focus for Benchmark Synthesis
  - Benchmark Generation Modeling Language
  - Modeling Avionics Scenario using BGML
- Concluding Remarks & Future Work

Concluding Remarks & Future Work

- In this presentation we described the design of CCMPerf, an open-source benchmark suite for CCM implementations
  - Applied CCMPerf to CIAO component middleware to quantify overhead
  - Next steps to benchmark Mico-CCM, Qedo
  - Have performance pages for these Component middleware implementations
- Services Benchmark include metrics for Real-time Event Channel integration into CIAO. www.dre.vanderbilt.edu/~gokhale/rtss_ec.pdf
- Domain Specific Benchmarks
  - Real-time CCM integration in CIAO

- **CCMPerf download from http://www.dre.vanderbilt.edu/Download.html**
- **BGML part of CoSMIC tool-suite http://www.dre.vanderbilt.edu/cosmic**