Extending Real-Time CORBA for Next-Generation Distributed Real-Time Mission-Critical Systems

Chris Gill and Ron Cytron
{cdgill, cytron}@cs.wustl.edu
www.cs.wustl.edu/~cdgill/omgrtws01.{ppt, pdf}
Center for Distributed Object Computing
Department of Computer Science
Washington University, St. Louis, MO

Wednesday, June 6, 2001

Work supported by Boeing, DARPA, and AFRL
Motivating Application

Boeing Bold Stroke Middleware Infrastructure Platform
- Used on CRAD, IRAD, and production systems
- Research conduit to production systems

Operations Well Defined
- Harmonic rates, bounded execution times
- Need criticality isolation assurances

Event Mediated Middleware Solution
- RT Enhanced TAO Event Channel
- Precedence DAG, scheduler per endsystem

Previous Generation Systems
- Fixed environment, static modes
- Used cyclic exec or RMS scheduling

Next Generation Systems
- Highly variable environment
- Large # of system states, dynamic modes
- Need dynamic & adaptive resource mgmt
- Need coordinated closed-loop QoS control
  - Across time-scales, system layers
  - *E.g.*, ACE+TAO, QuO, RT-ARM
Limitations With Existing Approaches

Historically, distributed and embedded RT systems built directly atop hardware/OS

- Tedious, error-prone, & costly over lifecycles

COTS middleware (e.g., CORBA) increasingly used to lower cost/time in real-world systems:

However, current COTS middleware lacks hooks for key domain-specific features, e.g.:

- Optimized integration w/ higher level managers
- Hybrid static-dynamic scheduling strategies
- Composition of scheduling strategies & dispatching mechanisms from primitive elements
- Adaptive domain-specific & run-time optimizations
Research Approach: the *Kokyu* Flexible Middleware Scheduling/Dispatching Framework

**Scheduler**
- Rate and priority assignment policy
- Rate propagation
- WCET propagation
- Tuple visitor
- Operation visitors
- Selected rates
- Propagated rates
- Rate tuples

**Dispatcher**
- Dispatching configuration
- RMS
- LLF
- Static rates
- Laxity
- Timers
- Mandatory
- Optional

**Application specifies characteristics**
- *e.g.*, criticality, periods, dependencies

**Scheduler assigns rates & priorities per topology, scheduling policy**
- Defines necessary dispatch configuration

**Dispatcher is (re)configurable**
- Multiple priority lanes
- *Queue, thread, timers* per lane
- Starts repetitive timers once
- Looks up lane on each arrival

**Implicit projection**
- Of specific scheduling policy into generic dispatch infrastructure
<table>
<thead>
<tr>
<th>Technical Challenge</th>
<th>Research Approach</th>
<th>Research Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>No one strategy optimal for every resource “niche”</td>
<td>Dispatching composed from primitive elements</td>
<td>Supports tailored “fit” of scheduling/dispatching</td>
</tr>
<tr>
<td>Co-scheduling resource managers and application</td>
<td>Decision lattice joining <em>a priori</em> analysis with empirical measurement</td>
<td>Allows run-time reflective and adaptive policy selection (in-progress)</td>
</tr>
</tbody>
</table>
Problem: Limitations with Existing Single-Policy Approaches

Optimal Heuristic Depends on Application-Specific Details:
- Example: RMS+LLF vs. MUF when rates are harmonic vs. non-harmonic
  – Feasibility vs. performance

Performance of Three Canonical Queue Ordering Disciplines
- Simple test with queue classes
- Randomly ordered enqueues
- Static $\rightarrow$ fixed sub-priority
- Deadline $\rightarrow$ time to deadline
- Laxity $\rightarrow$ time to deadline – WCET
- Enqueue overhead worse with $>$ load
- Overhead: static $<<$ deadline $<$ laxity
Solution: Composition of Scheduling Heuristics from Dispatching Primitives

Dispatcher

MUF

laxity

laxity

feasibility

static

static

static

laxity

RMS

+LLF

Gives Fine Grain Control over Feasibility / Performance Trade-Off

• With non-harmonic rates MUF may be feasible but RMS+LLF infeasible
• However MUF dispatching overhead is expected to be worse
  – Only 2 threads, but queue management/contention likely dominates
  – Mandatory - 1 laxity queue, optional - 1 laxity queue
• RMS+LLF performance is expected to be better, if feasible
  – 5 threads but greater fan-out of critical operations = lower contention
  – Mandatory – 4 static queues, optional - 1 laxity queue
Empirical Results: Tailored Policy Improves Deadline Success of Optional Operations

ASFD: Expectations from Theory and Measurement Confirmed
- Some improvement of RMS+LLF over MUF in practice
  - Made more optional operation deadlines under same overload conditions
- Lower overhead/queue & greater fan-out across queues in RMS+LLF
Co-Scheduling Resource Managers & Application

<table>
<thead>
<tr>
<th>Technical Challenge</th>
<th>Research Approach</th>
<th>Research Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>No one strategy optimal for each resource “niche”</td>
<td>Dispatching composed from primitive elements</td>
<td>Supports tailored “fit” of scheduling/dispatching</td>
</tr>
<tr>
<td>Co-scheduling resource managers and application</td>
<td>Decision lattice joining <em>a priori</em> analysis with empirical measurement</td>
<td>Allows run-time reflective and adaptive policy selection (in-progress)</td>
</tr>
</tbody>
</table>
Problem: Limitations with Existing Approaches to Co-Scheduling ARM and Application

Previously ad hoc

• Scheduled entire RT-ARM in a single priority lane
• However, RT-ARM is also divisible into mandatory and optional sets
  – Mandatory: could we adapt?
  – Optional: perform adaptation
• Key: mandatory + ARMm feasible
  – Or, no assurance of coherency
• Natural criticality partition over the set of all operations
  – Application mandatory
  – ARM mandatory
  – ARM optional
  – Application optional
• Given all this, we can do better
Solution: Use Empirical & A Priori Information to Co-Schedule Resource Mgrs & Applications

Preserve Invariant, but Optimize Performance
- Criticality: values partition ops for deadline isolation
- Definition: system schedulable if highest partition feasible
- Invariant: no lower partition can make a higher one infeasible
- Key: invariant strength – e.g., 1:1 criticality to priority over-constrains – Want safe optimizations

Decision Lattice (experiments in progress)
- A rich optimization space: topological? geometric?
- Spans criticality → prio/queue mappings
- E.g., over 4 partitions: {mandatory},{ARMm},{ARMo},{optional}
Empirical Evaluation
- Validates adaptive/hybrid scheduling approach
- Quantifies costs/benefits of discrete alternatives
- Powerful when combined with theoretical view
  - “Mining” technique for problems & properties

Composable Dispatching
- Enables domain-specific optimizations, especially when design decisions are aided by empirical data

Heuristic Space Experiments
- Will offer a quantitative blueprint for co-scheduling RT-ARM with OFP applications
- Will demonstrate a general co-scheduling technique where theory & empirical studies meet

Open-Source Code
- All software described here that is uniquely a part of my research will be made available in the ACE_wrappers distribution
  - First within TAO, then as a distinct Kokyu directory (summer 2001)