The Evolution of Real-Time CORBA:
A Brief Perspective on the Good, the Bad, and the Ugly

E. Douglas Jensen

jensen@real-time.org

http://www.real-time.org

Revised 7/15/03

MITRE

This is a brief retrospective on some of my perceptions about the evolution of Real-Time CORBA

- As Real-Time CORBA becomes more widely used, an increasing number of the participants at this event were not involved in the earlier days of its evolution ('95 on)
- This brief personal retrospective is intended to
  - provide historical perspective and rationale
  - expand your conceptual horizons
  - provoke you to learn more about certain conceptual issues in distributed real-time systems
- Remember: these are only my personal views; "YMMV"
- To protect the guilty, I will not mention the names of people or organizations

It is an old saying that "No one should watch sausage being made" – to which I add "or standards"

- Achieving consensus on any shared vision and on details can be slow (and not always enjoyable) for various reasons
- The RT-SIG (and even the submitter team) was (the SIG still is) composed of various factions (not entirely disjoint)
  - prospective real-time ORB vendors, vs. mainstream ORB vendors having business motivations to limit, or even prevent, the Real-Time CORBA standard (some were also on the AB)
    - customers who may not need it might make them (try to) supply it
  - traditional real-time vs. telecom
    - different views on priorities, scheduling, resource management
  - embedded vs. enterprise (e.g., DoD BM/C^2)
    - timeliness; static vs. dynamic; scale
  - a few researchers vs. a lot of practitioners (users, vendors)
    - standards work isn't supposed to be research

In each faction, “real-time” meant rather different things – particularly with respect to priorities

- Traditional real-time and distributed real-time
  - threads/tasks have static priorities (and other properties)
  - overloads engineered (or defined) away á priori
- Embedded
  - timeliness is often casual; priorities might not even be used
  - overloads engineered (or defined) away á priori
- Telecom
  - priorities are more of an application, than a system, concept – e.g., are applied to messages or resources instead of threads
  - overloads are frequent and must be handled gracefully
- Enterprise (e.g., DoD BM/C^2) real-time and distributed real-time
  - threads/tasks have dynamic priorities (and other properties)
  - overloads are frequent and must be handled gracefully

Example of real-time resource management for BM/C^2 in network-centric warfare: AMSTE

- Affordable Moving Surface Target Engagement (AMSTE)
- Netted GMTI sensors & shooter
- Moving/move-stop targets
- Maintain threat track from nomination through engagement, from tactically significant standoff ranges
- Provide precision fire control updates to weapons in flight
- Requires real-time orchestration of sensors, weapons, and data links

1 second deadline for sensor to tracker to missile
4 - 6 Hz update rate to missile

We had (and still have) to deal with the practitioner community's ambiguous real-time terms and concepts

- That community has many different, ambiguous, contradictory, and wrong definitions of the fundamental concepts of real-time computing (real-time, hard, soft, predictability, deterministic, etc.)
- Often, it falls back on the Justice Potter Stewart approach to controversial definitions –

  "*I can't define it [real-time, hard, soft, predictability, deterministic, etc.] but I know it when I see it*"
- That greatly impedes using "it" in the specifications
- Most of the real-time researchers weren't much help
- The RT-SIG largely wasted more than a year trying to write a multi-faction consensus-based and coherent
  - white paper for an RFI
  - RFP

  } big ugly Frankenstein monsters

MITRE

As unlikely as it may seem to some of you,
I'm not here today to tell you what *my* thoughtful definitions are

Four major decisions cleared the way to a Fixed Priority Real-Time CORBA RFP

- One important decision was when Jensen capitulated on the question of separate RFP's for
  - fixed priority scheduling (which led to the Real-Time CORBA 1 specification)
  - dynamic (e.g., deadline-based) scheduling (which led to the Real-Time CORBA 2 specification)
- Another was to write a separate RFP for Minimum CORBA (which was issued before the Fixed Priority Real-Time RFP)
- The third was providing priority mechanisms in a way that satisfied the needs of both traditional real-time and telecom real-time
- Fourth, a small representative group in the RT-SIG started over and produced a much better and smaller RFP for Fixed Priority Real-Time CORBA – as a separate compliance point

The RFP required the proposals to define "end-to-end predictability"

- The final unified proposal (and hence the spec) defines that *end-to-end predictability of timeliness in a fixed priority distributed system* means
  - respecting thread priorities between client and server for resolving resource contention during the processing of distributed invocations
  - upper bounding the duration of thread priority inversions during end-to-end processing
  - upper bounding the latencies of operation invocations

  (sometimes referred to as "Doug's Three Magic Bullets" of Real-Time CORBA 1)
- Unfortunately, this actually defined
  - how to achieve predictability in fixed priority systems
  - not what "predictability" is – e.g., what its metric is

The specification's priority mechanisms support both end-to-end predictability and control of resource allocation

- At least one priority inversion protocol (e.g., priority inheritance, priority ceiling protocol) in the ORB
  - also needed in the OS's
- Priority-based or non-multiplexed transport connections
- CORBA priority and mapping to/from node priorities
- Multiple server side, optionally priority laned, thread pools
- The choice between
  - propagating client's priority to the server, and respecting it in the ORB throughout the remote invocation cycle
    - an end-to-end timeliness necessity
  - handling server object invocations at those objects' priorities
    - a telecom necessity also useful for traditional real-time
- The CORBA mutex

MITRE

I assume this audience understands these mechanisms – except perhaps the reason for the CORBA mutex

The CORBA mutex is intended for constructing priority-respecting synchronizers for both the ORB and applications

- ORB implementations should use (almost) only synchronizers based on <u>this</u> mutex (not all do)
- Applications should use the CORBA mutex (almost always) instead of the OS synchronizers (most don't)
- Ideally, even the OS's and ORB (and dare I say NIC's?) would use the same mutex (none do – or will?) – c.f.
  - the Alpha and MK7.3a distributed real-time OS's
    - no OS/ORB dichotomy
  - DEC's "Libra" Alpha-based real-time ORB on VxWorks
    - applications and ORB instances and (hacked) OS instances all used only the ORB's synchronizers
- Realistically, not all resource management will employ priorities and mutexes consistently – so there will be some number and durations of end-to-end timeliness anomalies

Respecting priorities does not differentiate real-time from non-real-time ORB's (and systems)

- High degrees of priority compliance can be as valuable in non-real-time systems as in real-time ones –

  there, priorities are one of the mechanisms for managing resources to balance system throughput and job fairness
- The differentiation between real-time and non-real-time priority systems arises from real-time ones
  - having priority semantics based on application time constraints (deadlines being a simple special case)
  - employing the priorities to resolve resource contention with the objective of satisfying system timeliness criteria
    - how well all application time constraints are satisfied
    - and how predictable that satisfaction is
  - having predictability of timeliness as a correctness measure, not a performance one

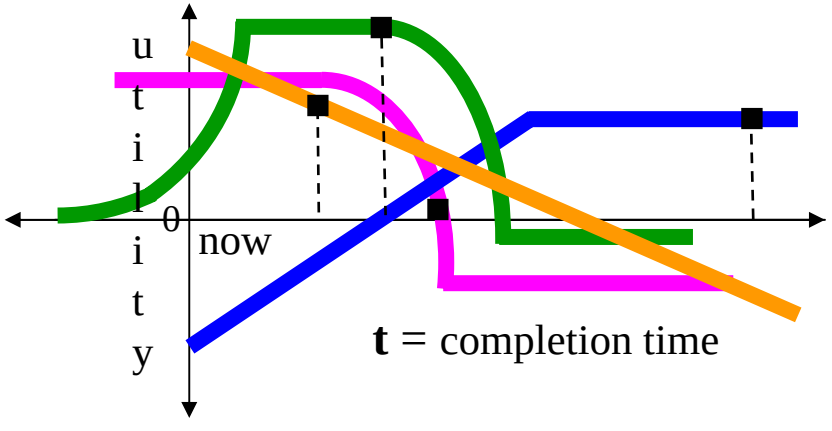
The Real-Time CORBA 1 (Fixed Priority) specification deliberately does not limit priority semantics to timeliness

- In that sense, instead of being called a "real-time ORB," it might more accurately be called "a distributed priority infrastructure that works"
  - if its priority mechanisms are used appropriately
  - (and better so if the OS's are priority respecting – not to mention the network…there are purpose-built ones that do)
- Thus, it could be valuable for non-real-time applications,

  just as traditional priority-based real-time OS's have proven to be valuable for certain non-real-time applications
- Maybe those mainstream CORBA vendors were right to be worried about "real-time" CORBA being a threat
  - except that the non-real-time world and real-time CORBA vendors don't get together much to exploit this opportunity
  - too bad, because the real-time market is pretty small

The Real-Time CORBA 2 (Dynamic Scheduling) specification generalizes the eligibility parameters

- It is intended to support arbitrary timeliness parameters and scheduling disciplines (with some help from an RTF)
- That specifically includes time/utility function time constraints and utility accrual scheduling disciplines
- A time/utility function expresses the utility to the system of an action's completion, as a function of its completion time



time/utility functions for ready threads

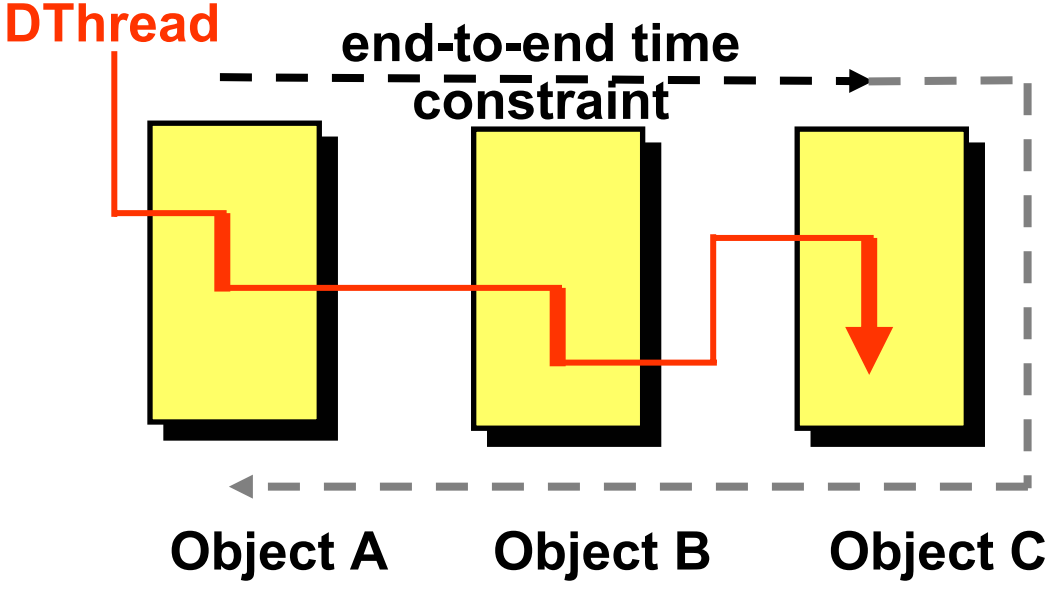
Schedule to maximize
$U = \sum u_i$ ■

MITRE

Time/utility functions and utility accrual disciplines were first (publicly) implemented in the Alpha OS

The Real-Time CORBA 2 specification also provides distributable threads as its schedulable entity

- The Real-Time CORBA 1.0 specification used the "activity" abstraction as an analysis/design concept for end-to-end predictability of timeliness, but it left the details of the abstraction undefined
- The Real-Time CORBA 2 specification defines *distributable threads* to provide a vehicle for end-to-end timeliness



MITRE

Distributable threads are based on distributed threads in the Alpha OS

Is Real-Time CORBA 2 "more real-time" in some sense than Real-Time CORBA 1? Or less?

- Real-time CORBA 2 directly supports
  - applications' actual time constraints, such as deadlines and time/utility functions
  - pluggable scheduling disciplines, such as EDF and UA

  while remaining backwards compatible with using priorities
- Real-time CORBA 2 supports generalized concepts of end-to-end predictability, and respect for eligibility end-to-end
- But arbitrary eligibility parameters could be those for (say) maximizing end-to-end throughput
- Other parameters could be propagated in support of other end-to-end properties, such as fault tolerance, security, etc.
- Hence, the Real-Time CORBA 2 specification could be valuable for non-real-time systems as well as for real-time ones – even more so than Real-Time CORBA 1

MITRE

Predictability of end-to-end security or reliability can be as critical as timeliness

As Paul Harvey says, “And now you know [some of] the rest of the story!”

- But this presentation was really intended in part to inspire the beginning of a new story for real-time CORBA

  to be written by the vendors and users