A Guided Tour of Real-time CORBA

OMG Real-time Workshop on Distributed Real-time Embedded Systems
July 2004

Presenters

Douglas Jensen
MITRE.

Shahzad Aslam-Mir
PrismTech Corporation, USA.

Chris Gill
Washington University, St Louis.

Irfan Pyarali
OOMWorks

A Guided Tour of Real-time CORBA

Agenda

Part 1: Introduction to concepts and terminology that distinguish between RT-CORBA

- 1.0 static, fixed priority, scheduling.
- 1.2 dynamic scheduling.

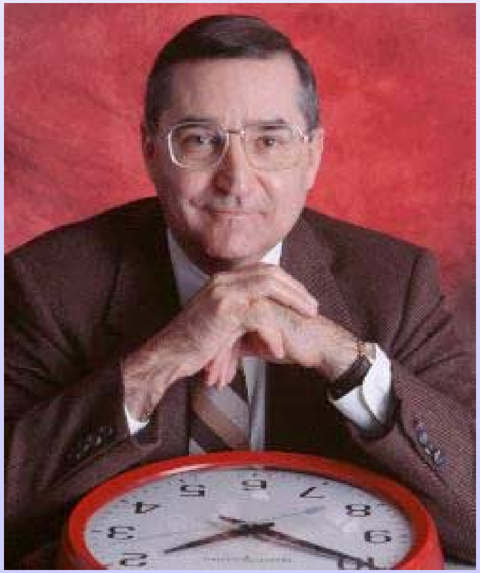
Part 2a: Using RT-CORBA in fixed priority statically scheduled systems.
Part 2b: Practical Implementation and Code examples.

Part 3a: Using RT-CORBA in dynamically scheduled systems.
Part 3b: Practical Implementation and Code examples.

Materials reproduced with the kind permission of Professor D.C. Schmidt, Dr I. Pyarali, Dr C. Gill, Dr D. Jensen Dr. S. Aslam-Mir and members of DOC research groups.

Real-Time CORBA Tutorial
Part 1

E. Douglas Jensen
Chief Scientist, D500
MITRE, Bedford, MA
jensen@{real-time | mitre}.org
www.{real-time | mitre}.org



A Guided Tour of Real-time CORBA

Part 1
Introduction to concepts and terminology that distinguish between RT-CORBA

1.0 static, fixed priority, scheduling
1.2 dynamic scheduling

Real-Time CORBA 1.0 and 1.2 have two major differentiations

- Scheduling
  - 1.0 is for static, fixed priority, systems
  - 1.2 is for dynamic scheduling systems
  - This distinction is the topic of this part of the tutorial
- Thread model
  - 1.0 provides for point-to-point timeliness using OS threads
  - 1.2 provides for end-to-end timeliness using ORB Distributable Threads
  - This distinction is the topic of Part 3 of the tutorial

Sequencing is one way of resolving contention for sequentially shared resources

- Common examples of sequentially shared resources in computing systems include
  - processors
  - networks
  - synchronizers
- Sequencing creates a sequence in which contenders access the sequentially shared resource – well-known examples include
  - priority-based sequence: highest priority first
  - deadline-based sequence: e.g., earliest deadline first
- (Constraint satisfaction is another way, commonly used for planning)

Some sequences are better than others for a given system, application set, and circumstance

- A resource access sequence is supposed to be chosen that
  - satisfies an access *eligibility optimality criterion* – e.g.,
    - meet all deadlines
    - minimize the number of missed deadlines
    - minimize mean tardiness
    - maximize accrued utility
  - subject to constraints – e.g.,
    - application-explicit access precedence
    - correctness – e.g., serializability (implicit access precedence)
    - resource capacity, compatibility, dependencies
  - acceptably well under the current circumstances

Satisfying access eligibility criteria “acceptably well” is measured in two dimensions

- Optimality of sequencing
  - i.e., how well all the time constraints of eligible actions are satisfied according to the optimality criterion
- Predictability of optimality of sequencing
  - i.e., how much can be known á priori about the optimality of sequencing
- An application or system is real-time *to the degree* that these two dimensions are
  - part of the application or system's logic
  - and thus correctness properties (vs. performance properties)

(note that this applies as much to soft as to hard real-time)

Informally, something is predictable to the degree that it can be known in advance

- Making a prediction involves using information with various kinds and degrees of uncertainties,
  to either deductively or inductively draw conclusions having various kinds and degrees of uncertainties
- In computing systems having time-constrained actions, the information needed for sequence timeliness predictions is about
  - the actions' properties, resource requirements, and dependencies
  - various aspects of the system's structure and behavior
  - the characteristics of the system's execution environment
- The conclusions are used for resource management to achieve the desired timeliness

Prediction is based on some particular model of the reality of interest

- The information used is generally incomplete and inaccurate and hence modeled in some way,
  and the predicting is performed according to the rules of the model
- The accuracy of the prediction depends on how well the modeled information and prediction process reflect the reality of interest
- The most common formal models for making predictions under uncertainty use classical probability theory to represent and manipulate uncertainties both in the observed or presumed information and in the predictions
- For example, consider analytical models for predicting performance (e.g., throughput) of non-time-constrained systems

The non-trivial nature of probability-based predictability is easily seen in a simple example

- The deterministic (or constant) distribution ($p = 1$ at $X = k$) is obviously the most predictable one
- Thus, it might seem intuitive that the least predictable distribution would be the uniform distribution ($p = P$ for all X)
- That intuition has been formally recognized since the 18th century as the *Principle of Indifference*
- But it leads to insoluble paradoxes which reveal it to be a heuristic that may be useful for suggesting hypotheses, rather than a logical principle
- However, the classical interpretation of probability does require that the Principle of Indifference be a logical principle
- This contradiction led to the subjective and frequentist interpretations of probability (and subsequently others)

An alternative intuitive perspective on the least predictable distribution in the context of classical probability theory

- Predictability of a probability density function is inversely proportional to its variability –
  as measured, for example, by its coefficient of variation
  c_v = variance/mean^2
- From that perspective, the deterministic distribution, whose $c_v = 0$, is still the most predictable
- But the standard form of the uniform distribution has a relatively low $c_v = 0.58$ and thus relatively high predictability
- Many distributions exist that have higher c_v's and are thus less predictable –

  for example, the extreme mixture of exponentials distribution has an arbitrarily large c_v and is thus arbitrarily unpredictable

A misconception in the field of real-time computing is the confusion of “predictability” with “deterministic”

- Predictability is a continuum
  - maximum predictability – called *deterministic* – is one end-point
  - the rest of the continuum is degrees of predictability (or non-determinism)
  - minimum predictability (or maximum non-determinism) is the other end-point
- A measure for predictability (or non-determinism), and characterization of the minimum predictability (maximally non-deterministic) end-point, depend on the particular predictability model –

  an example measure in probability models is coefficient of variation C_v

Optimality and predictability of optimality are orthogonal

- They generally must be traded off against one another according to application-specific requirements
  - one sequence may provide less optimality but with higher predictability
    - e.g., lower mean number of missed deadlines but lower variance in the number of missed deadlines
    - (a savings account provides a guaranteed low interest rate)
  - another sequence may provide more optimality but with lower predictability
    - e.g., higher mean number of missed deadlines but higher variance in the number of missed deadlines
    - (a mutual fund might provide a higher rate of return)
- Real-time systems emphasize predictability of timeliness optimality

Given the access eligibility criterion, a sequencing algorithm is chosen or devised to optimize it

- For example, well-known algorithms that are optimal – under each one's specific presumptions – for meeting all (hard) deadlines, include
  - rate monotonic
  - earliest deadline first
  - least laxity first
- (We will see later that the latter two disciplines are optimal for other criteria also)

Rate Monotonic Analysis

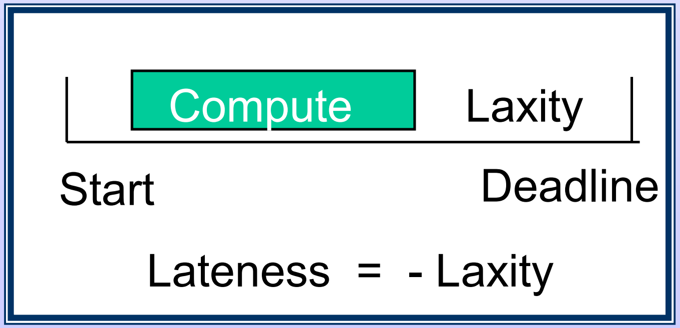
- A well-known discipline for assigning static priorities to periodic contenders is rate monotonic analysis (RMA)
  - priorities are a monotonic function of the rates – the higher the rate, the higher the priority
  - priorities may be adjusted by a priority inversion mechanism

Earliest Deadline First

- A well-known discipline for assigning dynamic priorities to periodic or aperiodic contenders is earliest deadline first (EDF)
  - priorities are assigned (not necessarily monotonically) in order of deadlines – the closer the deadline, the higher the priority
  - priorities may be adjusted by a priority inversion mechanism
- (Note that deadlines per se can be used directly without mapping them into priorities, if the sequencer permits, which is rare in real-time computing systems but common in other fields)
- Deadlines will be discussed further

Least Laxity First

- A well-known discipline for assigning dynamic priorities to periodic or aperiodic contenders is least laxity first (LLF)
  - priorities are assigned (not necessarily monotonically) according to laxity or slack time (deadline – execution time) – the lower the laxity, the higher the priority
  - priorities may be adjusted by a priority inversion mechanism



Pick an optimality criterion first then a discipline

- Often practitioners pick a sequencing discipline intuitively without first understanding what it optimizes vs. what they want optimized
- The archtypical example is the earliest deadline first discipline
  - when all deadlines can be met (given the necessary conditions), in both hard and soft real-time systems, it behaves intuitively
  - otherwise, its behavior is surprising to most real-time computing practitioners
    - it minimizes the maximum tardiness
    - which accesses will miss their deadlines is generally not known in advance

Sequencing may be either scheduling or dispatching

- *Scheduling* is deciding in what order <u>all</u> currently ready contenders will access a resource – i.e., creating a schedule of resource accesses
- *Dispatching* is deciding which is <u>the</u> most eligible ready contender when the resource access decision is made
- When scheduling is used, dispatching is vestigial – always the first contender in the schedule
- Dispatching without scheduling is sometimes more cost-effective than scheduling
- A sequencing optimality criterion and a corresponding algorithm are used for both
  - scheduling
  - dispatching without scheduling

Sequencing may be either static or dynamic

- Static sequencing is always static scheduling of static contenders in a static system context – a.k.a. “hard” real-time (the converse is not true)
  - all contenders are presumed to be known á priori , and to have á priori known, either fixed or bounded, properties such as
    - arrival rates
    - access durations
    - resource utilization dependencies and constraints
  - the schedule is constructed off-line
  - á priori scheduling analysis is possible in principle (but not always computationally tractable)
- Dynamic sequencing – scheduling and all dispatching
  - all other cases
  - In general, sequencing analysis is not possible á priori

Sequencing takes place at sequencing events

- A sequencing decision is required when certain events take place (depending on preemption or not), such as when
  - the currently accessing contender completes, blocks, is paused, is resumed, or is aborted
  - a (either newly arrived or extant) contender becomes ready
  - a contender's access eligibility parameters change
- If dynamic scheduling is being used, the scheduler creates a new schedule, and the most eligible contender is granted access to the resource
- If static scheduling is being used, the contender at the head of the schedule may be granted access to the resource
- If dispatching without scheduling is being used, the most eligible ready contender is granted access to the resource

Priority dispatching is the most frequently used form of sequencing in real-time computing systems

- It is often erroneously confused with, and referred to as, priority scheduling
- Most operating systems for both real-time and non-real-time computer systems do support actual priority scheduling as well as priority dispatching
- Priority scheduling and dispatching are not unique to real-time computing
  - the sequencing optimality criterion may be in terms of maximizing throughput, fairness, etc. instead of timeliness
- Priority dispatching or scheduling for real-time computing involves mapping time constraints (such as deadlines) to priorities, and selecting a timeliness optimality criterion
- Priority dispatching is dynamic (priority scheduling may be either static or dynamic)

Priority scheduling may be either static or dynamic – static scheduling is used for "hard" real-time

- Hard real-time as defined by the real-time computing research community (not by practitioners – they have various imprecise definitions based on latencies etc.)
  - the optimality criterion is "meet all deadlines"
  - the predictability of optimality is maximum – i.e., deterministic
  - it is required that all deadlines can be met, else the system has a design or operation error
    - consequences are application- or situation-specific – not necessarily a "disaster"
- Static priorities may be assigned in a variety of ways
- A well-known discipline for assigning static priorities to periodic contenders is <u>r</u>ate <u>m</u>onotonic <u>a</u>nalysis (RMA)
  - priorities are a monotonic function of the rates – the higher the rate, the higher the priority
  - priorities may be adjusted by a priority inversion mechanism

Priority scheduling may be either static or dynamic – dynamic scheduling is used for "soft" real-time

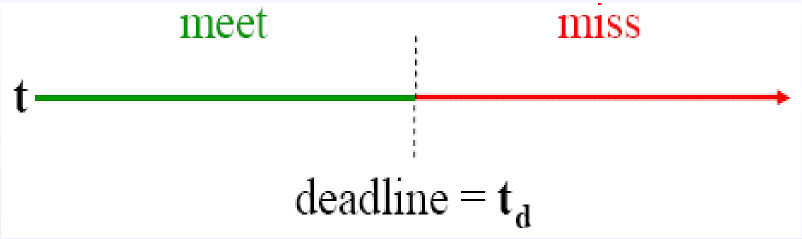
- Soft real-time – all the cases that are not hard real-time
  - any optimality criteria (e.g., minimize number of missed deadlines or mean tardiness, or maximize accrued utility)
  - any predictability of optimality, including deterministic
    - most of scheduling theory is about deterministic soft real-time
  - not "Que sera sera" – insufficient optimality or predictability of optimality may be as, or more, dangerous than in hard real-time – contrary to popular misconception
- Dynamic priorities may be assigned in a variety of ways – well-known disciplines include
  - earliest deadline first (EDF)
  - least laxity first

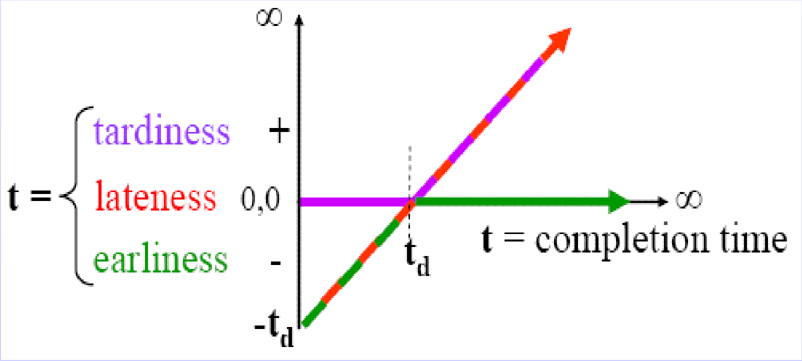
The earliest deadline first algorithm is used for scheduling and dispatching without scheduling

- EDF can be used either with deadlines directly or for assigning dynamic priorities derived from priorities
- EDF
    - optimal for meeting all deadlines when all deadlines can be met – always true in hard real-time, common in soft real-time
    - when not all deadlines can be met (common in soft real-time), it minimizes the maximum latency
        - which deadlines will be missed is not known in advance
        - often mis-represented in hard real-time cases as “instability”
- Those hard and soft criteria, and the EDF algorithm, are used for real-time systems (mostly outside of the computing context) for both
    - scheduling
    - dispatching without scheduling

Deadlines have major expressiveness disadvantages for dynamic scheduling

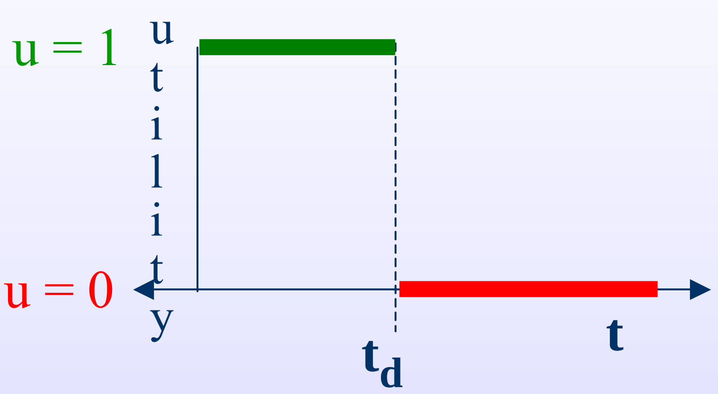
- Deadlines, as popularly spoken of in hard real-time computing, are only binary: an action either meets or misses it
- Scheduling theory deals with lateness, but deadlines have only
  - linear timeliness metric, *lateness* = completion time - deadline
  - single inflection point metric, *tardiness* = max[0,lateness]



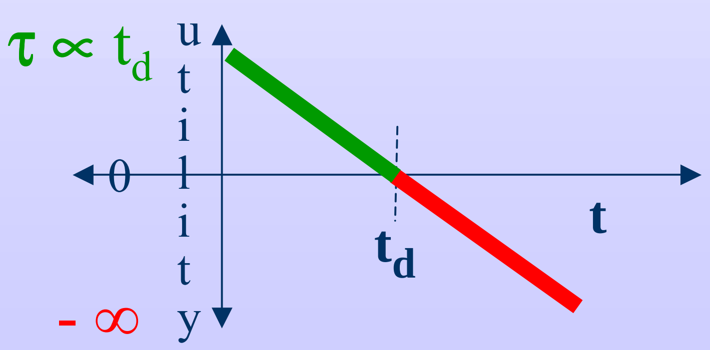


“Hard” deadlines and general deadlines can be represented by utility as a function of time

- Real-time computing’s “hard” deadline is a binary unit-valued downward step



- A general deadline in terms of negative lateness



Hard vs. soft deadlines are not defined with respect to the consequences of meeting them

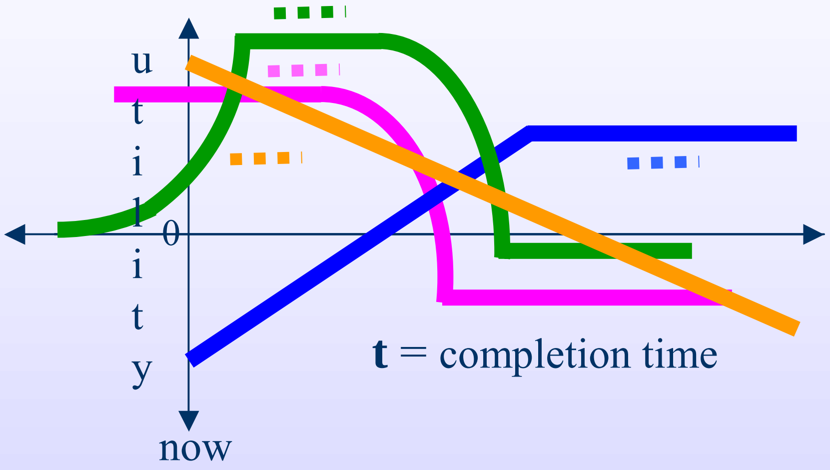
- The distinction between a hard and soft deadline is "syntactic" in the sense of being binary vs. linear
- The semantics of a deadline
  - i.e., the specific way in which system timeliness depends on whether any particular deadline is met, such as whether a miss constitutes a failure

  is not the correct distinction between a hard and soft deadline
- These semantics are properly specified as part of the sequencing optimality criteria (i.e., how deadlines are used for managing resources)

Deadlines can be generalized to more expressive *time/utility functions*

- Time/utility functions (TUF's)
  - express the utility to the system of completing a resource access as an application- or situation-specific function of when it completes
- TUF's typically are derived from the nature of the application
  - easily in the cases we have implemented
  - engineering tradeoffs can be made between expressiveness and complexity

Example

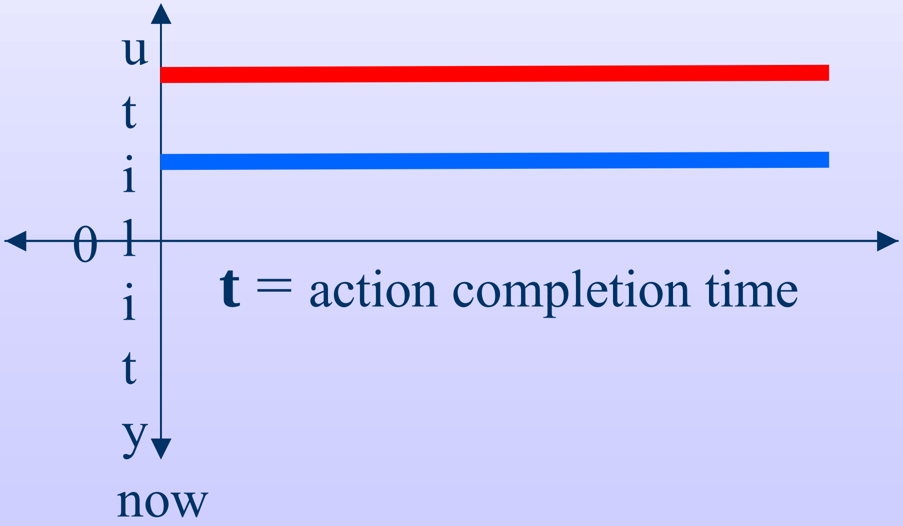


General time/utility functions

Expected or max execution times

Constant time/utility functions represent non-time-constrained actions

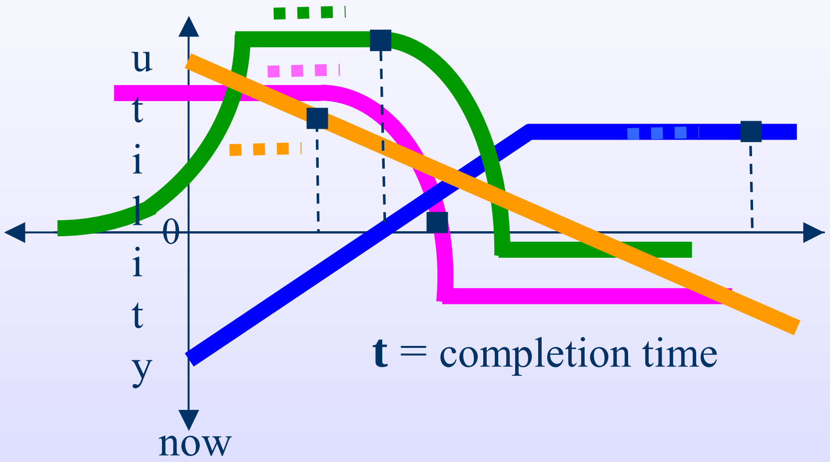
- Their utilities are one way of expressing their relative importance
- This has the advantage of allowing both time-constrained and non-time-constrained actions to be scheduled coherently by the same algorithm



The optimality criteria for sequencing with TUF's are based on accruing utility

- Utility accrual (UA) sequencing algorithms
  - sequencing activities according to optimality criteria based on
    - accruing utility – such as maximizing the sum of the utilities
    - plus satisfying dependencies such as resource constraints, etc.
- Can be either deterministic or stochastic
- Can provide utility bounds for individual accesses

Example



Expected or max execution time

■ Example scheduled completion times

Schedule to maximize
$U = \sum w_i u_i$ ■

Real-Time CORBA 1.0 and 1.2 differ in their support for sequencing

- Real-Time CORBA 1.0 is intended for static, fixed priority, systems
  - typically, smaller scale, lower level, subsystems
- Real-Time CORBA 1.2 is intended for dynamic systems, and allows for pluggable application-specific sequencing disciplines
  - over the entire range of system size and complexity, but most common in larger, more complex, systems