Real-Time Systems

Academic Year 2018/19
Master Degree in Computer Science
Department of Mathematics
University of Padua
Tullio Vardanega

Outline

Bibliography

- J. Liu, "Real-Time Systems", Prentice Hall, 2000
- A. Burns and A. Wellings, "Analysable Real-Time Systems - Programmed in Ada", Amazon Books, 2016
- State-of-the-art literature

1. Introduction

Initial intuition /1

- **Real-time system /1**
  - An aggregate of computers, I/O devices and *application-specific software*, characterized by
    - Intensive interaction with external environment
    - Time-dependent variations in the state of the external environment
    - Need to keep control over all individual parts of the external environment and to react to changes
  - System activities subject to timing constraints
    - Reactivity, accuracy, duration, completion, responsiveness: all dimensions of ***timeliness***
  - System activities inherently concurrent and increasingly parallel
  - The satisfaction of all system constraints must be proved

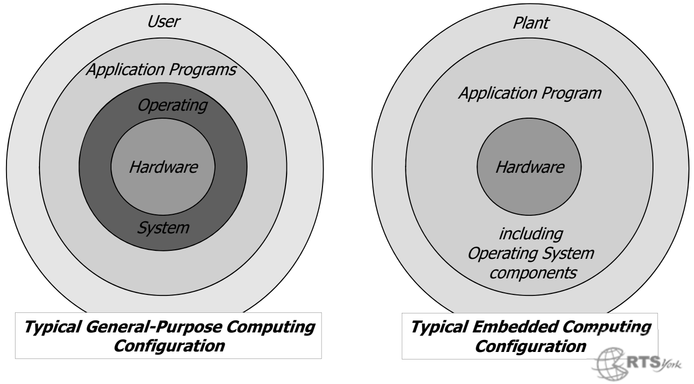
Initial intuition /2

- **Real-time system /2**
  - Operational correctness does not solely depend on the logical result but also on the time at which the result is produced
    - The computed response has an application-specific utility
    - Correctness is defined in the value domain and in the time domain
    - A logically-correct response produced later than due may be bad
- **Embedded system**
  - The computer and its software are fully immersed in an engineering system comprised of the external environment subject to its control

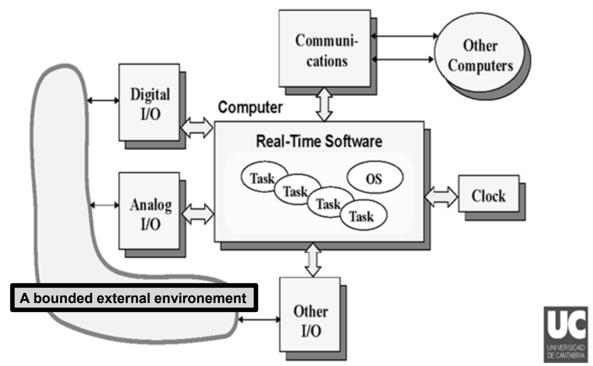
Initial intuition /3

- One key difference exists between **embedded systems** and **cyber-physical systems** (CPS), the new frontier of research in this domain
- Embedded systems are traditionally *closed* systems
  - The interaction with the environment is bounded and the system operation only varies within a fixed set of modes
- Cyber-physical systems are intrinsically *open*
  - Part of the environment is unknown
  - The functional needs may vary rapidly over time

Embedded system /1



Embedded system /2



Cyber-physical system

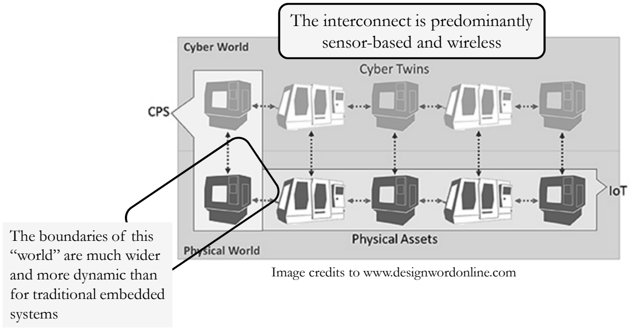


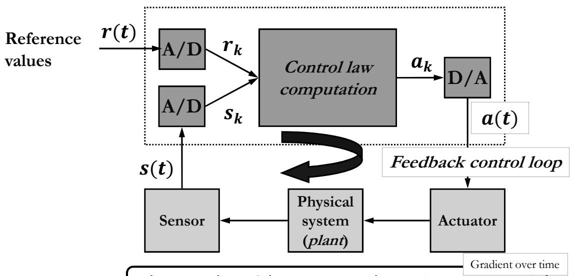
Image credits to www.designwordonline.com

Cybernetics: now and then

- Born in 1948 as the science of control systems
  - From the Greek κυβερνητης "steersman", which became "gubernator" in Latin
    - *Sensing* the external (physical) environment
    - *Computing* the distance from the expected status
    - *Actuating* devices that reduce that distance
  - Every control action performed on the external environment causes (positive or negative) *feedback*
  - The goal is to calibrate actions so that the system objectives is reached with bounded feedback

Example /1

- A digital system comprised of sensors and actuators



$$a_k = a_{k-2} + \alpha(r_k - s_k) + \beta(r_{k-1} - s_{k-1}) + \gamma(r_{k-2} - s_{k-2})$$

Example /2

- Factors of influence
  - Quality of response (***responsiveness***)
    - Sensor sampling is typically periodic with period T
      - For the convenience of control theory
    - Actuator commanding is produced at the time of the next sampling
      - As part of feedback control mathematics
    - System stability degrades with the width of the sampling period
  - Plant ***capacity***
    - Good-quality control reduces oscillations
    - A system that needs to react rapidly to environmental changes and is capable of it within $rise\ time\ R$ requires higher frequency of actuation and thus faster sampling → hence shorter T
    - A rule-of-thumb R/T ratio normally ranges [10 .. 20]

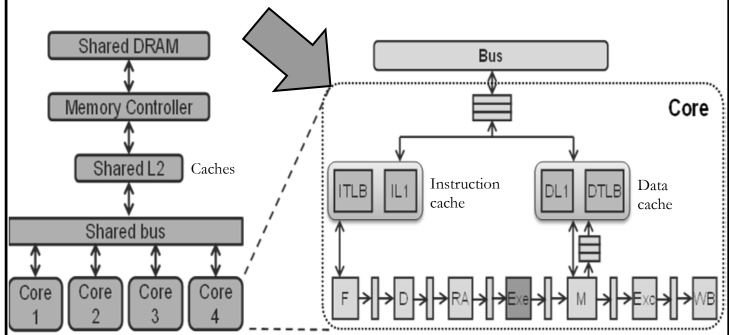
Application requirements /1

- A control system consists of (possibly distributed) resources governed by a real-time operating system
- The RTOS design must meet stringent ***reliability*** requirements
- Measured in terms of *maximum acceptable probability of* ***failure***
  - Typically set in the range $10^{-10} .. 10^{-5}$ per unit of operational life/service time (hour / run)

Application requirements /2

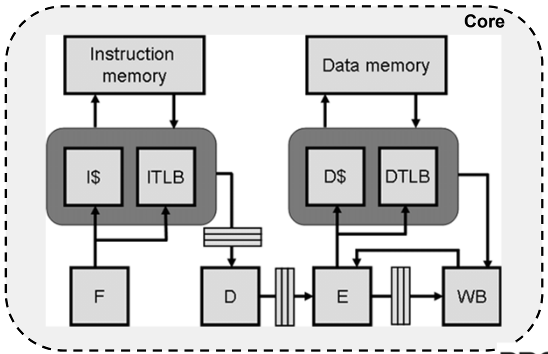
- Safety-critical systems
  - E.g., Airbus A-3X0: 10^{-9} probability of allowable system failure per hour of flight
    - One failure in 10^9 hours of flight ($> 114^+k$ years!)
- Business-critical real-time systems
  - E.g., satellite system: between 10^{-6} and 10^{-7} probability of allowable failure per hour of operation
    - One failure in 10^7 hours of operation (about 1,141 years!)

Understanding the hardware /1



Courtesy of PROXIMA

Understanding the hardware /2



Courtesy of PROXIMA

Key characteristics /1

- **Complexity**
  - In algorithms, mostly because of the need to apply discrete control over analog and continuous physical phenomena
  - In development, mostly owing to more demanding verification and validation processes
- **Heterogeneity** of components and of processing activities
  - Multi-disciplinary engineering (spanning control, SW, and system)
- Extreme **variability** in size and scope
  - From tiny and pervasive (nano-devices) to very large (aircraft, plant)
  - In all cases, finite in computational resources
- Proven ***dependability***

Key characteristics /2

- Must respond to events triggered by the external environment as well as by the passing of time
  - Double nature: *event-driven* and *time-driven*
- Continuity of operation
  - The whole point of a real-time embedded system is that it must be capable of operating without (constant) human supervision
  - Nearly no keyboard-based interaction!
- Software architecture inherently concurrent
- Must be temporally ***predictable***
  - Need for static (off-line) verification of correct temporal behavior
  - How does that relate to ***determinism***?

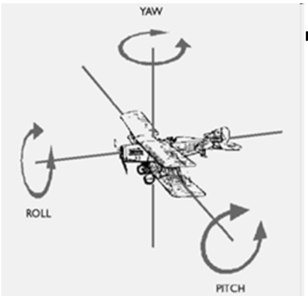
Predictability and determinism

- Predictability (what can be known a priori) may be regarded as a continuum
  - Its maximum end-point is deterministic a-priori knowledge (absolute certainty)
  - Its minimum end-point is total absence of a-priori knowledge (see what happens …)
- Seeking predictability implies reasoning about kinds and degrees of uncertainty
  - Very rarely we have full a-priori knowledge

Meeting real-time requirements

- Minimizing the average response time of application tasks is the goal for general-purpose computing but it is not for RTS!
- "Real-time computing is **not** equivalent to fast computing" [Stankovic, 1988]
- Given real-time requirements and a HW/SW implementation, how can one show that those requirements are met?
  - Testing and simulation are not enough
  - Maiden flight of space shuttle, 12 April 1981: there was a $\frac{1}{67}$ probability for a *transient overload* occurring at initialization; it never did in testing; it did at launch
- System-level ***predictability*** is what we need
  - Central to it, is knowing the *worst case*

Example /3



Any three-dimensional rotation can be described as a sequence of $roll\ (x)$, $pitch\ (y)$, $yaw\ (z)$ rotations (Euler angles)

- Complex systems must support multiple distinct periods T_i
  - Easier to set a **harmonic** relation between all T_i
    - This removes the need for concurrency of execution in the relevant computations
    - But it causes coupling between possibly unrelated control actions which is a poor architectural choice
  - There may be diverse components of speed
    - *Forward, side slip, altitude*
  - As well as diverse components of rotation
    - *Roll, pitch, yaw*
  - Each of them requires separate control activities each performed at a specific rate

Example /4

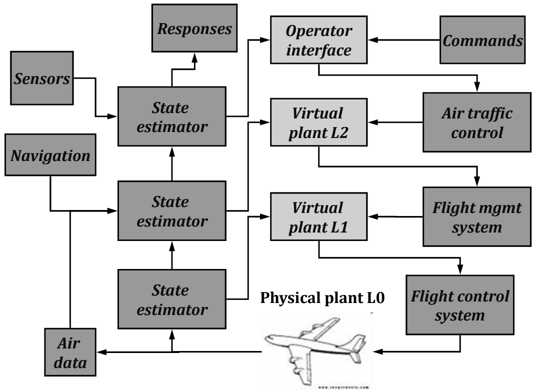
(Artificially) harmonic multi-rate system

- 180 Hz cycle
  - Check all sensor data and select sources to sample
  - Reconfigure system in case of read error
- 90 Hz cycle (at every 2nd activation)
  - Perform control law for pitch, roll, yaw (internal loop)
  - Command actuators
  - Perform sanity check
- 30 Hz cycle (at every 6th activation)
  - Perform control law for pitch, roll, yaw (external loop) and integration
- 30 Hz cycle (at every 6th activation)
  - Capture operator keyboard input and choice of operation model
  - Normalize sensor data and transform coordinates; update reference data

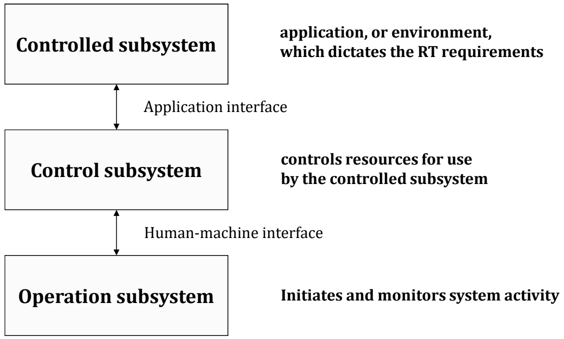
Example /5

- Command and control systems are often organized in a hierarchical fashion
  - At the lowest level we place the digital control systems that operate on the physical environment
  - At the highest level we place the interface with the human operator
    - The output of higher-level controllers becomes a reference value $r(t)$ for lower-level controllers
  - The more composite the hierarchy the more complex the interdependence in the logic and timing of operation

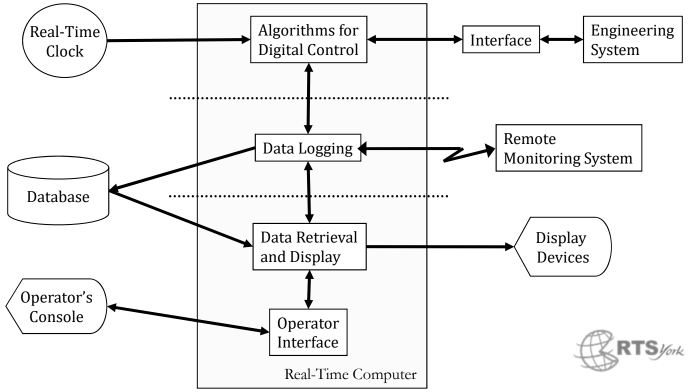
Example /6



A conceptual model



A typical embedded system



An initial taxonomy /1

- The prevailing classification stems from the traditional standpoint of control algorithms
- **Strictly periodic** systems
  - Harmonic multi-rate (artificially harmonized)
  - Polling for not-periodic events
- **Predominantly** (but not exclusively) **periodic** systems
  - Lower coupling
  - Better responsiveness to not-periodic events
- **Predominantly not-periodic systems** but still predictable
  - Events arrive at variable times but within bounded intervals
- **Not-periodic and unpredictable** systems
  - Another ballgame!

Some terminology

- ***Time-aware***
  - A system that makes explicit reference to wall-clock time
    - E.g., open vault door at 9.00 AM
- ***Reactive***
  - A system that must produce outputs within *deadlines* relative to specific (input) events
- Control systems are reactive by nature
  - Hence required to constrain the time variability (***jitter***) of their input and output
    - Input jitter and output jitter control

Definitions in the SW domain /1

- ***Job***
  - Unit of work selected for execution by the scheduler
  - Needs physical and logical *resources* to execute
  - Each job has an entry point where it awaits activation
- ***Task***
  - Unit of functional and architectural composition
  - Issues jobs (one at a time, until completion) to perform actual work
    - One such task is said to be *recurrent*

An initial taxonomy /2

- ***Periodic*** tasks
  - Their jobs become ready at regular intervals of time, T
  - Their arrival is synchronous to some time reference
- ***Aperiodic*** tasks
  - Recurrent but irregular
  - Their arrival cannot be anticipated (asynchronous)
- ***Sporadic*** tasks
  - Their jobs become ready at variable times but at bounded minimum distance from one another

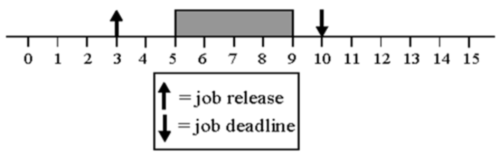
Definitions /2

- ***Release time***
  - When a job should become eligible for execution
    - The corresponding trigger is called ***release event***
    - There may be some temporal delay between the arrival of the release event and when the scheduler actually recognizes the job as ready
  - May be set at some *offset* from the system start time
    - The offset of the first job of task τ is named ***phase***, φ, and it is one of the attributes of τ

Definitions /3

- ***Deadline***
  - The time by which a job must complete its execution
  - May be < (*constrained*), = (*implicit*), > (*arbitrary*) than the next job's release time
- ***Response time***
  - The time span between the job's release and its actual completion
- The longest admissible response time for a job j_i is termed the job's *relative deadline,* D_i
- The algebraic summation of release time and relative deadline is termed *absolute deadline,* d_i

Example



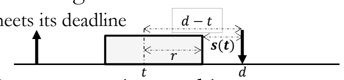
Job is released at time 3.
It's (absolute) deadline is at time 10.
It's relative deadline is 7.
It's response time is 6.

Definitions /4

- ***Hard*** *deadline*
  - If the consequences of a job completing past the deadline are serious and possibly intolerable
  - Satisfaction must be demonstrated off line
- ***Soft*** *deadline*
  - If the consequences of a job occasionally completing past the assigned deadline are tolerable
  - The quantitative interpretation of "occasional" may be established in either probabilistic terms or as a ***utility function***

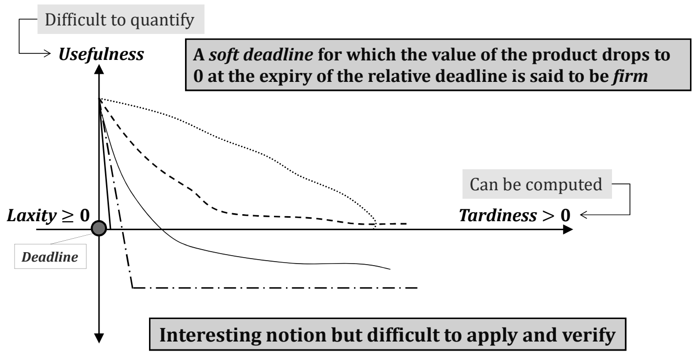
Definitions /5

- ***Laxity*** (aka ***slack***)
  - $s(t) = (d - t) - r$ defines the *slack* $s(t)$ at time t of job J with deadline d and remaining time of execution r
    - A job with non-negative laxity meets its deadline



- ***Tardiness***
  - The distance between a job's response time and its deadline
    - A job with negative laxity has tardiness
- ***Usefulness***
  - Value of (residual) utility of the job's computational product as a function of its tardiness

Utility function



An initial taxonomy /3

- According to timing requirements
  - **Hard real-time** (HRT) tasks
    - Whose jobs have hard deadlines
  - **Soft real-time** (SRT) tasks
    - Whose jobs have soft deadlines
  - **Firm real-time** (FRT) tasks
    - Whose jobs have soft deadlines but usefulness ≤ 0 past the deadline
  - **Not real-time** tasks
    - Do not exhibit timing requirements
- This taxonomy extends to real-time systems
  - Which however are mixed in nature

Abstract models /1

- Resources
  - ***Active*** (processor, server)
    - They "do" what they have to
      - Execute machine instructions, move data, process queries, etc.
    - Jobs *must* acquire them to make progress toward completion
    - Active resources have a *type*
      - Those of the same type can be used interchangeably by a job
      - Those of different types cannot
        - Processors may have different speed, which has major impact on the rate of progress for the jobs that run on them

Abstract models /2

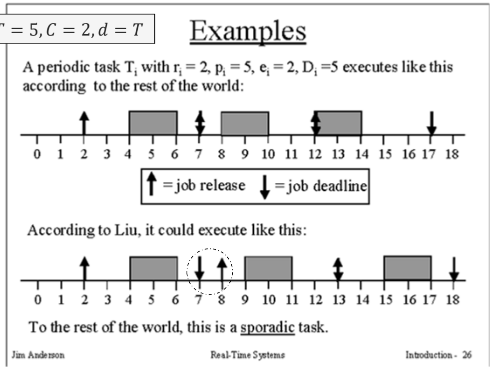
- Resources
  - ***Passive*** (memory, shared data, semaphores, …)
    - They don't do anything per se
      - Jobs *may need* some of them to do what they have to
    - They may be reused if use does not exhaust them
    - If always available in sufficient quantity to satisfy all needs, they are said to be *plentiful* and can be ignored
    - Passive resources that matter to real-time systems are those that may cause *bottlenecks*
      - Access to memory may matter more (owing to *arbitration*) than memory itself (which may be considered plentiful)

Abstract models /3

- Temporal parameters
  - ***Jitter***
    - Variability in the release time or in the time of input (data freshness) or output (stability of control)
  - ***Inter-arrival time***
    - Separation between the release time of successive jobs which are not strictly periodic
      - Job is *sporadic* if a guaranteed minimum such value exists
      - Job is *aperiodic* otherwise
  - ***Execution time,*** C
    - For any job J_i, C_i may vary between a *best-case* (BCET) C_i^b and a *worst-case* (WCET) C_i^w

Periodic task and sporadic task

$\varphi = 2, T = 5, C = 2, d = T$



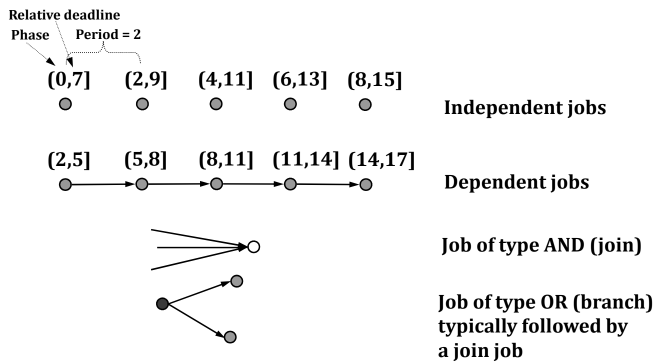
Abstract models /4

- ***Periodic model***
  - Comprises periodic and sporadic jobs
  - Accuracy of representation decreases with increasing jitter and variability of execution time
  - ***Hyperperiod*** H_S of task set $S = \{\tau_i\}, i = 1, \dots, N$
    - Defined as LCM (least common multiple) of task periods $\{T_i\}$
  - ***Utilization***
    - For every task τ_i : defined as the ratio between execution time and period : $U_i = \frac{C_i}{T_i} \leq 1$
    - For the system (*total utilization*) : $U = \sum_i U_i \leq m$, where m is the number of CPUs ($m = 1$, for now)

Abstract models /5

- Fixing execution parameters
  - The time that elapses between when a periodic job becomes ready and the next period T is certainly $< T$
  - Setting phase $\varphi > 0$ and deadline $D < T$ for a job may help limit its output jitter (**why?**)
  - The jobs of a system may be independent of one another
    - Hence they can execute in any order
  - Or they may be subject to *precedence constraints*
    - As it is typically the case in collaborative architectural styles
    - E.g., producer – consumer

Extended precedence graphs (task graphs)



Precedence constraints

- One job's release time cannot follow that of a successor job
- ***Effective release time (ERT)***
  - For a job J_i with predecessors $\{J_{k=1,\dots,i-1}\}$, ERT_i this is the *latest* value between its own release time and the maximum effective release time of its predecessors, ERT_k, plus C_k
- One job's deadline cannot precede that of a predecessor job
- ***Effective deadline (ED)***
  - For a job J_i with successors $\{J_{k=i+1,\dots,n}\}$, ED_i is the *earliest* value between D_i and the minimum effective deadline of its successors, ED_k, less C_k
- For single processors with preemptive scheduling, we may disregard precedence constraints and just consider ERT and ED

Abstract models /6

- Fixing design parameters
  - Permissibility of job preemption
    - May depend on the capabilities of the execution environment (e.g., ***non-reentrancy***) but also on the programming style
    - Preemption causes time and space overhead
  - Job ***criticality***
    - May be assimilated to a priority of execution eligibility
    - In general indicates which activities must be guaranteed possibly even at the cost of others
  - Permissibility of resource preemption
    - Some resources are intrinsically preemptable
    - Others do not permit it

Which ones?

Abstract models /7

Recall
BCET: best-case execution time
WCET: worst-case execution

- Selecting jobs for execution
  - The scheduler assigns a job to the processor resource
  - The resulting assignment is termed ***schedule***
- A schedule is ***valid*** if
  - Each processor is assigned to at most 1 job at a time
  - Each job is assigned to at most 1 processor at a time
  - No job is scheduled before its release time
  - The scheduling algorithm ensures that the amount of processor time assigned to a job is $\geq$ than its BCET and $\leq$ than its WCET
  - All precedence constraints in place among tasks as well as among resources are satisfied

Abstract models /8

- A *valid schedule* is said to be ***feasible*** if it satisfies the temporal constraints of every job
- A *job set* is said to be ***schedulable*** by a scheduling algorithm if that algorithm always produces a valid schedule for that problem
- A *scheduling algorithm* is ***optimal*** if it always produces a feasible schedule when one exists
- Actual systems may include multiple schedulers that operate in some hierarchical fashion
  - E.g., some scheduler governs access to logical resources; some other schedulers govern access to physical resources

Abstract models /9

- Two algorithms are of prime interests for real-time systems
  - The *scheduling algorithm*, which we should like to be *optimal*
    - Comparatively easy problem
  - The *analysis algorithm* that tests the *feasibility* of applying a scheduling algorithm to a given job set
    - Much harder problem
- The scientific community, but not always in full consistency, divides the analysis algorithms in
  - ***Feasibility tests***, which are exact
    - Necessary and sufficient
  - ***Schedulability tests***, which are only sufficient

Further characterization /1

	Time-Share Systems	Real-Time Systems
Capacity	High throughput	Ability to meet timing requirements: Schedulability
Responsiveness	Fast average response	Ensured worst-case latency
Overload	Fairness	Stability of critical part

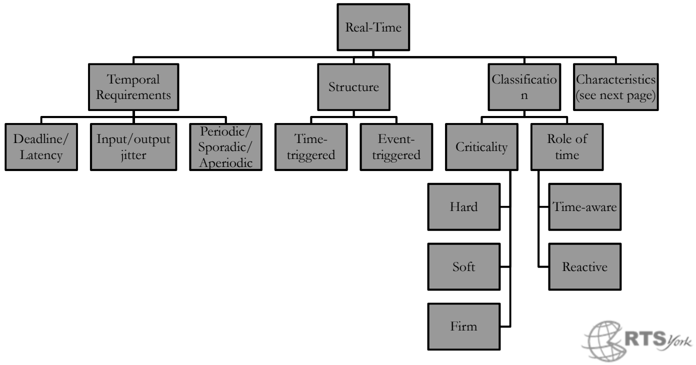
Further characterization /2

- The design and development of a RTS mind the worst case before considering the average case (if at all)
  - Improving the average case is of no use and it may even be counterproductive
    - The cache addresses the average case and therefore operates *adversarially* to the needs of real-time systems
- Stability of control prevails over fairness
  - The former concern is selective the other general
- When feasibility is proven, starvation is of no consequence
  - The non-critical part of the system may even experience starvation

Summary /1

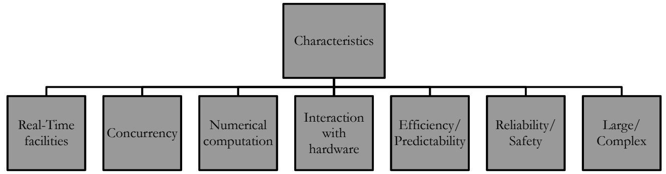
- From initial intuition to more solid definition of real-time embedded system
- Survey of application requirements and key characteristics
- Taxonomy of tasks
- Dispelling false myths
- Introduced abstract models to reason in general about real-time systems

Summary /2



2018/19 UniPD – T. Vardanega Real-Time Systems 53 of 537

Summary /3



2018/19 UniPD – T. Vardanega Real-Time Systems 54 of 537