Real-Time Systems

Anno accademico 2015/16
Laurea magistrale in informatica
Dipartimento di Matematica
Università di Padova
Tullio Vardanega

1. Introduction

Outline

Bibliography

- J. Liu, "Real-Time Systems", Prentice Hall, 2000
- A. Burns and A. Wellings, " "Concurrent and Real-Time Programming in Ada", Cambridge University Press

Initial intuition /1

- **Real-time system – I**
  - An aggregate of: computers, I/O devices, and application-specific software; all characterized by
    - Intense 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 are inherently concurrent
  - The satisfaction of such constraints must be proved

Initial intuition /2

- **Real-time system – II**
  - 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 function
    - Correctness is defined in the value domain and in the time domain
    - A logically-correct response produced later than due may be as bad as a wrong response
- **Embedded system**
  - The computer and its software are fully immersed in an engineering system comprised of the external environment subject to its control

A look into the future

- One key difference exists between **embedded systems** and **cyber-physical systems** (CPS), the new frontier of research in this domain
- Embedded systems are essentially *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

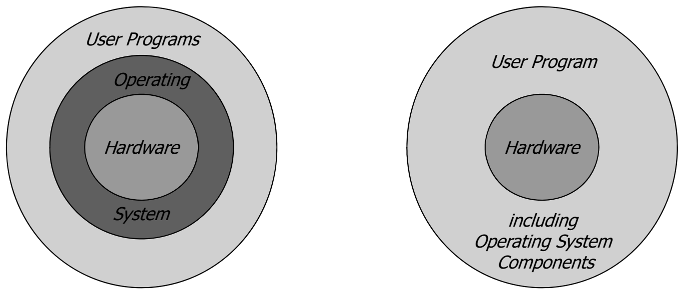
Application requirements /1

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

Application requirements /2

- Safety-critical systems
  - E.g., Airbus A-3X0: 10^{-10} probability of failure per hour of flight
    - One failure in 10^{10} hours of flight (about 11.5 million years!)
- Business-critical real-time systems
  - E.g., satellite system: between 10^{-6} and 10^{-7} probability of failure per hour of operation
    - One failure in 10^{7} hours of operation (about 11,306 years!)

Embedded system



Typical General-Purpose Computing Configuration

Typical Embedded Computing Configuration

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
  - In multi-disciplinarity (spanning control, software, and system engineering)
- 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 *clock- (or 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
- Software architecture is inherently concurrent
- Must be temporally ***predictable***
  - Need for static (off-line) verification of correct temporal behavior
  - Not easy at all

False myths to dispel /1

- Real-time systems design is empirical and not scientific
  - *False* : we shall see much of that in this class
- The increase in CPU power shall satisfy timing requirements coming from software of any sort
  - *False* : we continue to observe lateness all around us
- The essence of real-time computing is speed
  - *False* : we are interested in predictability, not speed
- The real-time systems discipline is no other than performance engineering
  - *False* : we shall here what it is made of

False myths to dispel /2

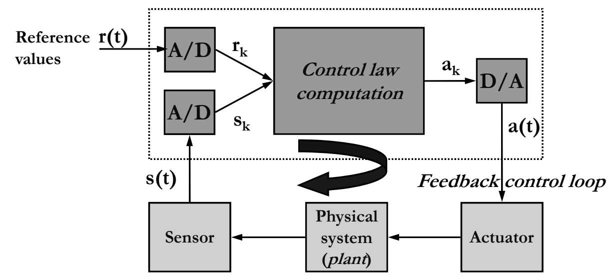
- Real-time programming is low-level
  - *False* : verification is so much easier if programming is higher-level
- All real-time “problems” have long been solved in other areas of computer science
  - *False* : operation research solves (possibly similar) problems with probabilistic and/or one-shot techniques
  - *False* : general-purpose computer science in general addresses average-case optimizations

Meeting real-time requirements

- It is <u>not</u> sufficient to minimize the average response time of application tasks
  - "Real-time computing is not equivalent to fast computing" [Stankovic, 88]
- Given a set of demanding real-time requirements and an implementation based on fast HW and SW, how can one show that those requirements are met?
  - Surely not only via testing and simulation
  - Counter-evidence: maiden flight of space shuttle, 12 April 1981: 1/67 probability that a *transient overload* occurs during initialization; and it actually did!
- System-level ***predictability*** is what we need

Example /1

- A digital system of sensors and actuators

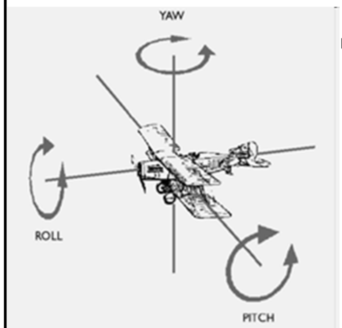


$$a_k = a_{k-2} + \alpha(r_k - s_k) + \beta(r_{k-1} - s_{k-1}) + \gamma(r_{k-2} - s_{k-2})$$

Example /2

- Factors of influence
  - Quality of response (**responsiveness**)
    - Sensor sampling is typically periodic (for convenience)
    - 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 period T
    - A “good” R/T ratio ranges [10 .. 20]

Example /3



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

- Complex systems must support multiple distinct periods T_i
  - It is convenient 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

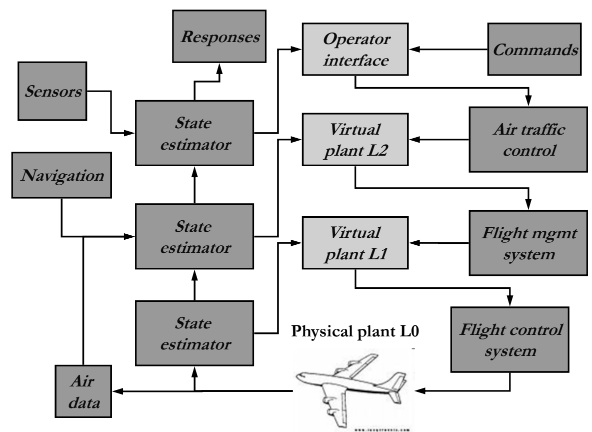
Example /4

- 180 Hz cycle (*harmonic multi-rate* functions)
  - 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

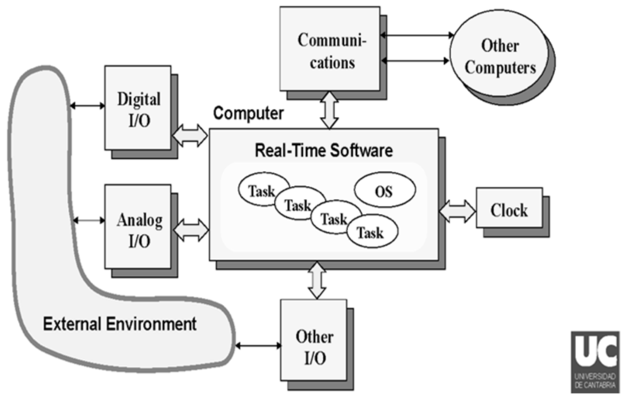
Example /5

- 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 high-level controller becomes a reference value **r(t)** for some low-level controller
  - The more composite the hierarchy the more complex the interdependence in the logic and timing of operation

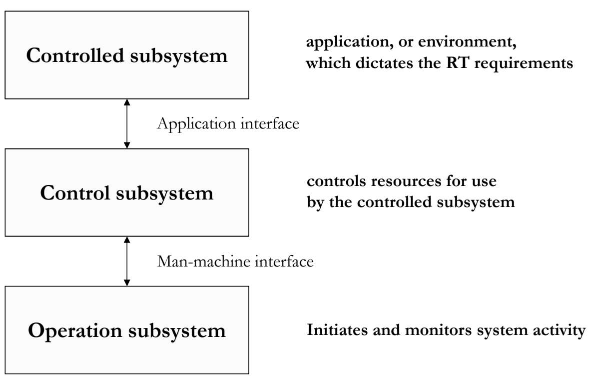
Example /6



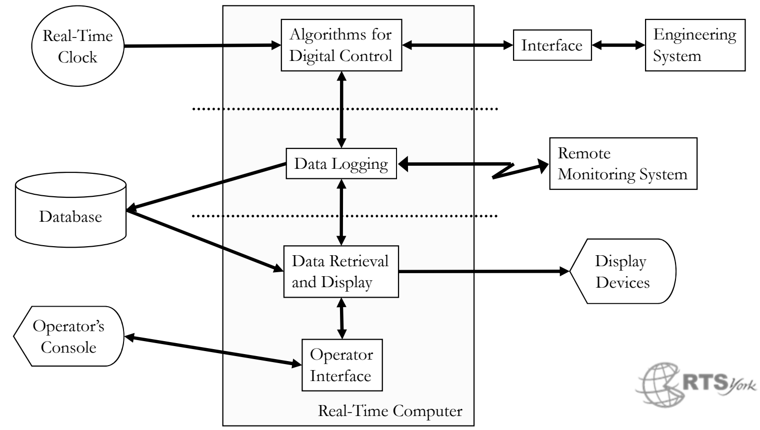
An overall vision



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 time
    - E.g., open vault door at 9.00 AM
- *Reactive*
  - A system that must produce outputs within deadlines relative to inputs
- 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 /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) to perform actual work
    - One such task is said to be *recurrent*

An initial taxonomy /2

- ***Periodic*** tasks
  - Their jobs become ready at regular interval of time
  - 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

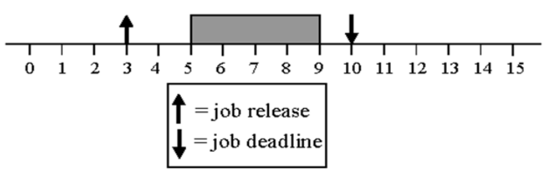
Definitions /2

- ***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*** and it is an attribute of τ

Definitions /3

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

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.

Jim Anderson Real-Time Systems Introduction - 18

Definitions /4

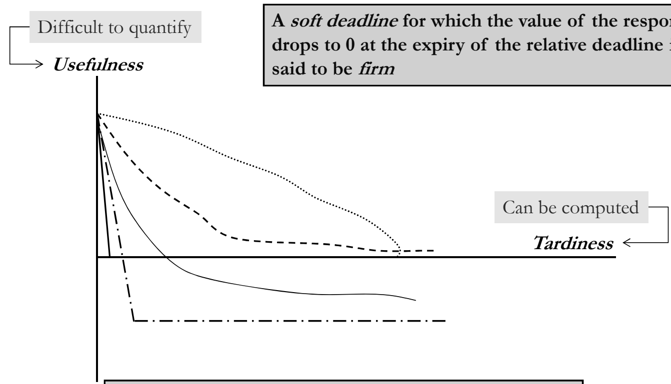
- *Hard deadline*
  - If the consequences of a job completing past the deadline are serious and possibly intolerable
    - Satisfaction must be demonstrated off line
- *Soft deadline*
  - If the consequences of a job completing past the assigned deadline are tolerable as long as the violation event is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

Definitions /5

- *Tardiness*
  - The temporal distance between a job's response time and its deadline
    - Evaluates to 0 for all completions *within* deadline
- *Usefulness*
  - Value of utility of the job's computation product as a function of its tardiness
  - Normally associated to the notion of ***laxity***
    - The slack $s(t)$ at time t of a job J with deadline d and remaining time of execution r is $s(t) = (d - t) - r$

Utility function

A *soft deadline* for which the value of the response drops to 0 at the expiry of the relative deadline is said to be *firm*



Interesting notion but difficult to apply and verify

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 obviously has major impact on the rate of progress of jobs

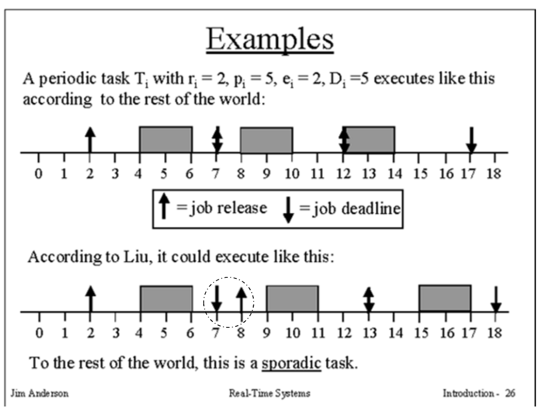
Abstract models /2

- Resources
  - *Passive* (memory, shared data, semaphores, …)
    - May be reused if use does not exhaust them
    - If always available in sufficient quantity to satisfy all requests they are said to be *plentiful* and are excluded from the space of the problem
    - To make progress, jobs may *need* some of them, together with active resources
    - 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 value exists
      - Job is ***aperiodic*** otherwise
  - ***Execution time***
    - May vary between a *best-case* (BCET) and a *worst-case* (WCET)

Periodic task and sporadic task



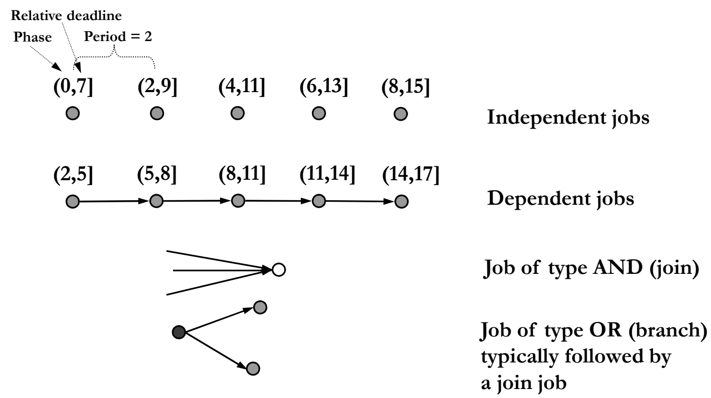
Abstract models /4

- ***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 $S = \{\tau_i\}, i = 1, \dots, N$
    - LCM (least common multiple) of periods $\{T_i\}$
  - ***Utilization***
    - For every task τ_i : ratio between execution time and period : $U_i = \frac{C_i}{T_i}$
    - For the system (*total utilization*) : $U = \sum_i U_i$

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 jitter in its response time (why?)
  - The jobs of a system may be independent of one another
    - Hence they can execute in any order
  - Else 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)



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 incurs 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 (**which ones?**)
    - Others do not permit it
      - Which becomes one of the four preconditions to deadlock

Abstract models /7

- Selecting jobs for execution
  - The scheduler assigns a job to the processor resource
    - Notice we are talking single core here
  - 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 no less than its BCET and no more 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 the temporal constraints of every job are all satisfied
- 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* that 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

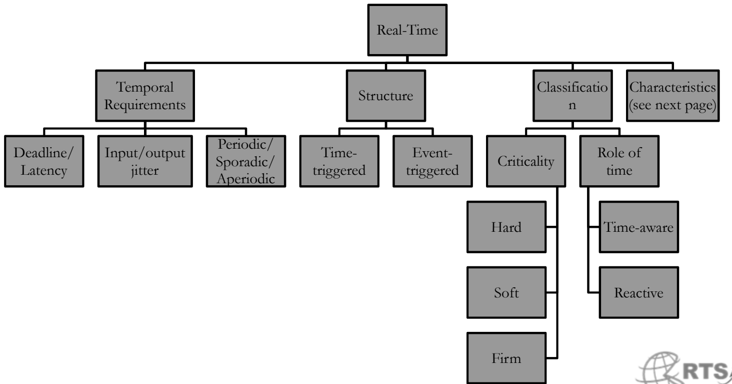
Further characterization /2

- The design and development of a RTS are concerned with the worst case as opposed to the average case
  - Improving the average case is of no use and it may even be counterproductive
    - The cache addresses the average case and therefore operates according to a counterproductive principle for 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



Summary /3