Real-Time Systems

Anno accademico 2009/10
Laurea magistrale in informatica
Dipartimento di Matematica Pura e Applicata
Università di Padova
Tullio Vardanega

Outline

- **Introduction**
- **Dependability issues**
- **Scheduling issues**
- **More on fixed-priority scheduling**
- **Task interactions and blocking**
- **System issues**
- **Considering distribution**
- **Bibliography**

1. J. Liu, "Real-Time Systems", Prentice Hall, 2000
2. A. Burns, A. Wellings, "Concurrent and Real-Time Programming in Ada", Cambridge University Press, 2007
3. A. Burns, A. Wellings, "Real Time Systems and Programming Languages: Ada 95, Real-Time Java and Real-Time C/POSIX", Addison-Wesley, 2009

Introduction

Initial intuition – 1

- **Real-time system – I**
  - An aggregate of computers, I/O devices and specialized software, all characterized by
    - Intensive interaction with external environment
    - Time-dependent variations in the state of (parts of) the external environment
    - Need to keep (software) control over all individual parts of the external environment and to react to changes
  - System activities subject to timing constraints
    - Reactivity (responsiveness), accuracy, duration, completion
  - 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 (algorithmic, functional) result but also on the time at which the result is produced
    - The computed response has a *utility function* that depends on the application
    - Correctness is logical and temporal
  - A logically-correct response produced later than expected may be as bad a 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

Application requirements – 1

- Control subsystem consists of possibly distributed resources governed by an RTOS (*real-time operating system*)
- RTOS design must meet critical reliability requirements
  - Typically measured in terms of **Maximum Acceptable Probability of Failure** (ranging 10^{-10} to 10^{-5})

We shall return to the interpretation of this term

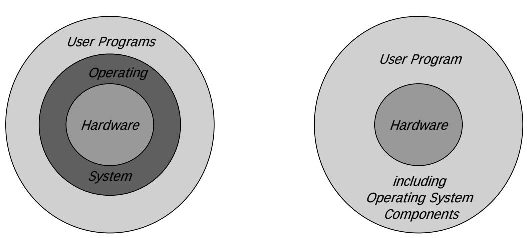
Application requirements – 2

- Safety-critical systems
  - E.g., Airbus A-320: 10^{-10} probability of failure per hour of flight
- Business-critical real-time systems
  - E.g., satellite system: between 10^{-6} and 10^{-7} probability of failure per hour of operation

Real-time system

- Obligation
  - To produce logically correct results within pre-assigned deadlines
- Computational correctness
  - Encompasses both **logical** and **timing** correctness
    - That is, computational correctness in both
      - ***Value domain***
      - ***Time domain***

Embedded system



Typical General-Purpose Computing Configuration

Typical Embedded Computing Configuration

Key characteristics – 1

- **Complexity**
  - Algorithmic, mostly because of the need to apply discrete control over analog (continuous) physical phenomena
  - Development, mostly owing to more demanding verification and validation processes
- **Heterogeneity** of components and of processing activities
  - Multi-disciplinary (control, software, and system engineering)
- Extreme **variability** in size and scope
  - From very tiny and pervasive (nano-devices) to very large (an aircraft)
  - 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, preventive) verification of correct temporal behavior
    - Not easy at all

We shall return to the interpretation of this term

False myths – 1

- The design of real-time systems 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 – 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 more often addresses average-case optimizations

Meeting real-time requirements

- It is not sufficient to minimize the average response time of each application task
  - "Real-time computing is not equivalent to fast computing" [Stankovic88]
- Given a set of demanding RT 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
  - E.g. (maiden flight of space shuttle, 12 April 1981): 1/67 probability that a transient overload occurred during initialization; and it did, in spite of all the testing done
- System predictability is what we need

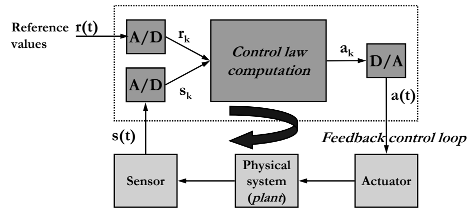
Predictability

- Crucial property of a real-time system
  - Functional and timing behavior of the system must be as **deterministic** as necessary to meet the real-time specifications
- Two general paradigms for the design of predictable RTOS
  - **Event-Triggered** (ET)
    - System activity initiated in response to the occurrence of specific events
  - **Time-Triggered** (TT)
    - System activity initiated at predefined instants of a globally synchronized clock

We shall return to the interpretation of this term

Example – 1

- Digital system of sensors and actuators

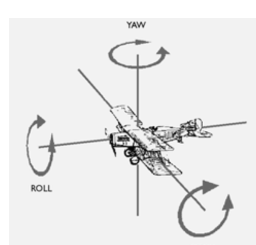


$$a_k = a_{k-2} + \alpha(r_k - s_k) + \beta(r_{k-1} - s_{k-1}) + \gamma(r_{k-2} - s_{k-2})$$

Example – 2

- Factors of influence
  - Quality of **responsiveness**
    - Sensor sampling is typically periodic
    - 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" ratio R/T ranges [10 .. 20]

Example – 3



- Complex systems have to support multiple distinct periods T_i
  - It is usually convenient to set a **harmonic** relation between all T_i
    - This however incurs undesirable coupling between possibly unrelated control actions
  - There may for example be diverse components of speed
    - *Forward, side slip, altitude*
  - As well as diverse components of rotation
    - *Roll, pitch, yaw*
  - This requires a score of control activities each performed at a specific rate

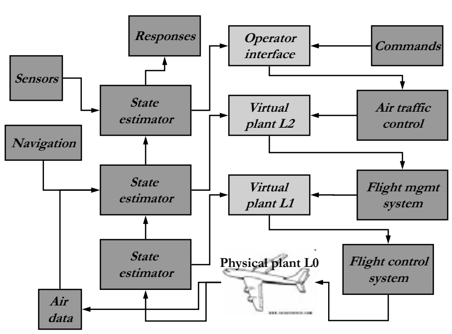
Example – 4

- Flight control (harmonic multi-rate)
  - 180 Hz cycle
    - Check all sensor data and select sources to sample
      - In case of reading error: reconfigure system
    - 30 Hz cycle (at every 6^{th} activation)
      - Capture operator keyboard input and choice of operation model
      - Normalize sensor data and transform coordinates
      - Update reference data
    - 30 Hz cycle (at every 6^{th} activation)
      - Perform control law for pitch, roll, yaw (external loop)
      - Perform integration
    - 90 Hz cycle (at every 2^{nd} activation)
      - Perform control law for pitch, roll, yaw (internal loop)
    - Perform control law yaw (internal loop) on outputs of 90 Hz cycle
    - Command actuators
    - Perform sanity check

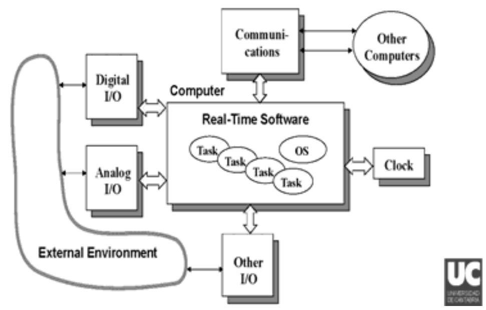
Example– 5

- 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

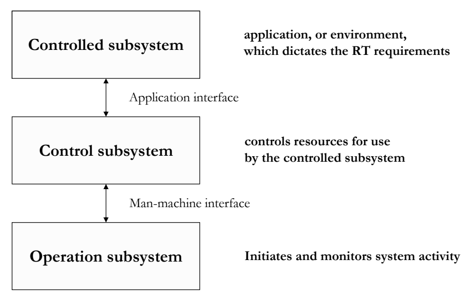
Example – 6



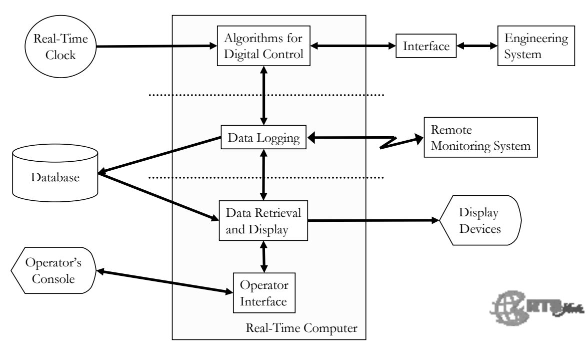
An overall vision



A conceptual model



A typical embedded system



An initial taxonomy – 1

- The prevailing (traditional) classification stems from the viewpoint 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 output within deadline relative to input
    - Control systems are reactive
    - 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
- **Task**
  - Unit of functional and architectural composition
  - Issues jobs (one at a time) to perform actual work
- **Release time**
  - When a job should become ready (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 job is actually recognized as ready by the scheduler
  - May be set at a given distance (offset) from the system start time
    - The offset of the first job of task `T` is named *phase* and it is an attribute of `T`

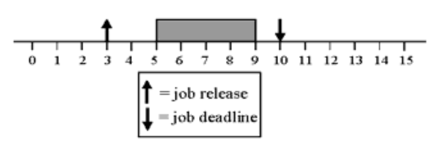
An initial taxonomy – 2

- **Periodic** (or synchronous) tasks
  - Become ready at regular interval of time
- **Aperiodic** (or asynchronous) tasks
  - Are recurrent but irregular
  - Their execution time cannot be anticipated
- **Sporadic** tasks
  - Become ready at variable but bounded time intervals

Definitions – 2

- **Deadline**
  - The time by which a job must complete its execution
    - For example, by the next release time
  - In general it is a fully arbitrary value and may be <, =, > 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



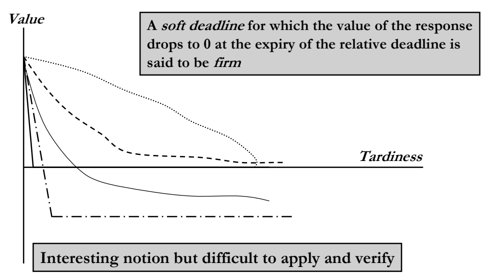
Job is released at time 3.
It's (absolute) deadline is at time 10.
It's relative deadline is 7.
It's response time is 6.

Jim Anderson Real-Time Systems Introduction - 18

Definitions – 3

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

Utility function



Tasks classification

- According to timing requirements
  - **Hard real-time** (HRT) tasks
    - Tasks whose deadlines are hard
  - **Soft real-time** (SRT) tasks
    - Tasks whose deadlines are soft deadlines
  - **Non real-time** (NRT) tasks
    - Tasks that do not exhibit real-time requirements

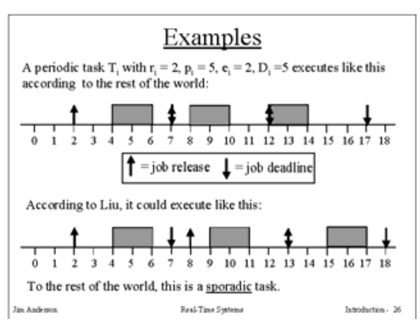
Classes of real-time systems

- **Hard real-time** (HRT) systems
  - Subclass of real-time systems in which all, most or some (not isolated) tasks have hard deadlines
- **Soft real-time** (SRT) systems
  - Subclass of real-time systems in which no deadlines are hard
- **Firm real-time** (FRT) systems
  - Equivalent to SRT except that there is no benefit from late delivery of service

Abstract models – 1

- Resources
  - Active (*processor*, *server*)
  - 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
- Temporal parameters
  - **[Release-time] Jitter**
    - Possible variability in the release time
  - **Inter-arrival time**
    - Separation time between the release time of successive jobs that are not strictly periodic
      - **Sporadic job** if a guaranteed minimum value exists
      - **Aperiodic job** otherwise

Periodic task and sporadic task



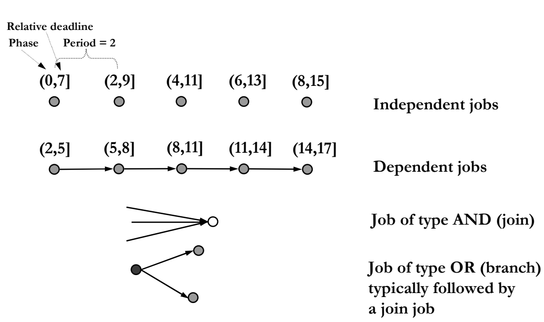
Abstract models – 2

- Execution time
  - May not be a constant and can vary between a *best-case execution time* (BCET) and a *worst-case execution time* (WCET)
- **Periodic model**
  - Comprised of periodic and sporadic jobs
  - Accuracy of representation decreases with increasing jitter and variability of execution time
  - *Hyperperiod* H of task set $\{Ti\}_{i=1,..,N}$
    - LCM (least common multiple) of periods {Pi}
  - *Utilization*
    - For every task Ti : ratio between execution time and period : Ui = Ei / Pi
    - For the system (***total utilization***) : ∑i Ui

Abstract models – 3

- Fixing execution parameters
  - The time that elapses between when a periodic job becomes ready and the next period P is certainly < P
  - Setting phase > 0 and deadline D < P for a job may help limit jitter in its response time
  - 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 – 4

- Fixing functional parameters
  - Permissibility of job preemption
    - May depend on the capabilities of the execution environment
    - But also on the programming style
      - *Non-reentrancy*
    - Preemption incurs non-null time overhead
  - Job *criticality*
    - May be assimilated to a priority of execution eligibility
    - In general indicates which activities must be guaranteed even possibly at the cost of others
  - Permissibility of resource preemption
    - Some resources are intrinsically preemptable (luckily! Which ones?)
    - Others do not permit it
      - Which becomes one of the four preconditions to deadlock

Abstract models – 5

- Selecting job for execution
  - The scheduler assigns a job to the processor resource
  - The resulting assignment is termed *schedule*
  - A schedule is *valid* if
    1. Each processor is assigned to at most 1 job at a time
    2. Each job is assigned to at most 1 processor at a time
    3. No job is scheduled before its release time
    4. 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
    5. All precedence constraints in place among tasks as well as among resources are satisfied

Abstract models – 6

- 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
- In an actual system there may be multiple schedulers that operate in some hierarchical fashion
  - Some scheduler govern access to logical resources
  - Some other schedulers govern access to physical resources

Abstract model – 7

- Two algorithms are of prime interests for real-time systems
  - The scheduling algorithm, which we should it was optimal
    - Comparatively easy problem
  - The algorithm to compute **feasibility analysis**
    - Much harder problem
- For the scientific community, but not always in fully consistency
  - **Feasibility tests** are exact
    - They are necessary and sufficient
  - **Schedulability tests** 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 real-time system 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 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

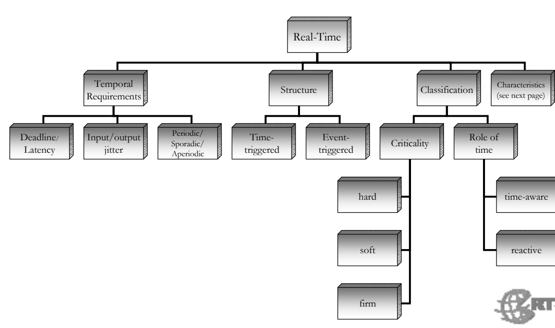
Meeting reliability requirements

- **Fault avoidance** techniques based on
  - Quality control
  - Robust engineering of components
    - However, cost penalty for engineering reliability into components through reduced failure rate
- **Fault tolerance** techniques based on
  - Use and management of redundant components
    - Made possible by microprocessor technology as weight, volume and power requirements associated with redundant hardware decreased

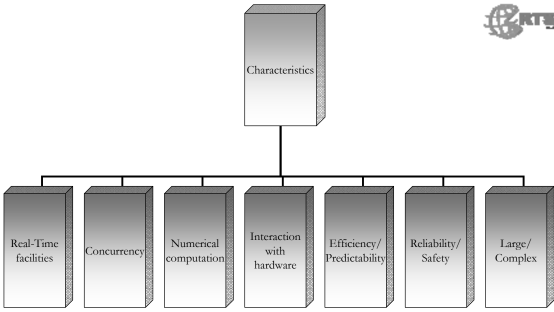
Summary – 1

- 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


Characteristics
Real-Time facilities
Concurrency
Numerical computation
Interaction with hardware
Efficiency/ Predictability
Reliability/ Safety
Large/ Complex