Design of Real-Time Software

Reference model

Reinder J. Bril
Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
System Architecture and Networking Group
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
r.j.bril@tue.nl, http://www.win.tue.nl/~rbril

January 2014
Model

- Model (of a system):
  - Abstraction (of that system)
    - leaving out details irrelevant to a given set of criteria
    - preserving the properties of interest
- (Scope) Our reference model
  - explicitly addresses relevant issues in real-time systems (in particular of the controlling software)
  - ...but must be mapped eventually onto an execution environment
    - OS, hardware, run-time system, ...
Events, responses and tasks

- **Event**: state change
  - usually at the RTCS boundary, under control of the environment, or from a timer or a task
  - an event indicates a state change that requires a **response**

- **Recording** external events:
  - **Informing** (the controlling software);
    - *event-driven*
    - typically: changes in discrete environmental variables
      - hazardous condition, packet passes a sensor
  - **Sampling** (by the controlling software):
    - *time-driven*
    - typically: continuous external values
      - temperature, humidity, pressure
Events, responses and tasks (cnt’d)

- **Task**: the sequence of actions that must be carried out to respond to an event
- Recurring events → perform task repeated
  - Task instance, task execution or *job*: an instance of the action sequence of a task
    - we can speak about the $k^{th}$ instance of task $i$
    - an event generates a job (“releases a task”) ...often 1-1, sometimes 1-$n$ correspondence
- Tasks must be *scheduled* appropriately such as to let *instances* meet their deadlines
- Tasks can be *time-driven* or *event-driven*
Task attributes

- A task has
  - a *name* (the $i^{th}$ task)
  - a *(worst-case)* computation time $C_i$
  - a *(relative, worst-case)* deadline $D_i$
  - a *(worst-case)* period, sometimes $T_i$
  - a *phase* (start of job 0)
- A job $\tau_{i,k}$ has
  - an *activation* (or *release*) time $a_{i,k}$
  - an *(absolute, worst-case)* deadline $d_{i,k}$
  - a *begin* (or *start*) time $b_{i,k}$
  - a *finalization* (or *end*) time $f_{i,k}$
Task deadlines

- **(Worst-case) Deadline:** the maximum time before a job must complete
  - *Relative:* the value $D$ to be added to the activation time $a$
  - *Absolute:* the result $d$ of the addition

- The value associated with the execution of a job after the deadline determines the *type* of deadline
Deadline types

- **Soft**
  - A response is still valuable after the deadline, but value decreases steadily after that.
    - Example: interaction with human users. People get impatient.

- **Firm**
  - A response has no value after the deadline.
    - Example: a video frame that cannot be shown in time can be skipped.

- **Hard**
  - Damage is done if a response does not come in time.
Derived attributes

- \( R_{i,k} = f_{i,k} - a_{i,k} \)  
  response time
- \( U_i^{\tau} = C_i / T_i \)  
  utilization for task \( \tau_i \)
- \( U = \sum U_i^{\tau} \)  
  total utilization of task set
- \( H = \text{lcm} (T_1, T_2, \ldots) \)  
  hyperperiod
- \( L_{i,k} = f_{i,k} - d_{i,k} \)  
  lateness
- \( E_{i,k} = \max \{0, L_{i,k}\} \)  
  tardiness (or exceeding time, i.e. active after deadline)
- \( X_{i,k} = D_i - C_i \)  
  laxity (or slack time)
Task (event) types – arrival patterns

- (Strictly) Periodic
  \[ a_{i,k} = \phi_i + k \times T_i \]

- Periodic with Activation Jitter \(AJ_i\)
  - redefinition of \(\phi_i\): the start of the arrival interval of job 0
  \[ \phi_i + k \times T_i \leq a_{i,k} \leq \phi_i + k \times T_i + AJ_i \]

- Sparse or sporadic: minimum inter-arrival time \(T_i\)
  \[ a_{i,k} + T_i \leq a_{i,k+1} \]

- Elastic: min. and max. inter-arrival times \(WT_i\) and \(BT_i\)
  \[ a_{i,k} + WT_i \leq a_{i,k+1} \leq a_{i,k} + BT_i \]
Job (task instance) constraints

- **Precedence**
  - *enforced* limitations on order (producer/consumer relationships)
    - AND-type (wait for a whole set), OR-type (wait for one of a set)
    - typical example: only single task instance active
  - generally: wait for a *condition* on the state

- **Resource sharing**
  - mutual exclusion

- **Data dependency**
  - *implied* limitations on data usage, e.g. ordering
    - (unsynchronized producer/consumer through buffering)

- **Temporal dependency**
  - limitation on time differences between job completions
    - e.g. synchronizing audio/video
Scope of the model

- The overall task properties come from the system specification
  - external events, response times
- All subsequent details represent design
  - decomposition of tasks
  - internal concurrency, specific implementations
  - choice of preemptivity,
- ... and are driven by
  - an understanding of the building blocks where the task-set is to be mapped upon
    - i.e., the resource model (model of the execution environment)
      - processor(s), memory, size and number of resources, provided OS-type services, ....
      - pre-emptivity of resources
  - the need to find a feasible schedule
    - e.g. a task might need decomposition to make it schedulable