AADL : about scheduling analysis
1. Issues about real-time scheduling: AADL to the rescue
2. Focus on fixed-priority scheduling:
   - Basics on uniprocessor
3. AADL components/properties to scheduling analysis
4. An example with Cheddar
Real-Time scheduling theory

1. **A set of tasks models** (to model functions of the system)
2. **A set of analytical methods** (feasibility tests)
   - E.g. Worst Case Response Time

   \[ R_i \leq \text{Deadline} \]

   \[ R_i = C_i + \sum_{j \in h_p(i)} \left( \frac{R_i}{P_j} \right) \cdot C_j \]

3. **A set of scheduling algorithms**: build the full scheduling/GANTT diagram
Real-Time scheduling theory is hard to apply

- Real-Time scheduling theory
  - Theoretical results defined from 1974 to 1994: feasibility tests exist for uniprocessor, periodic tasks, shared resources
  - Extension through simulation for other cases
- Now supported at a decent level by POSIX 1003 RTOS, ARINC653, …

- Industry demanding
  - Yet, hard to use
Real-Time scheduling theory is hard to apply

- Feasibility tests not always exist for modern architectures
  - Multi-cores, distributed, asynchronous, hierarchical

- Requires strong theoretical knowledge
  - Numerous theoretical results: how to choose the right one?
  - Numerous assumptions for each result.
  - How to abstract/model a system to access schedulability? (e.g. task dependency)

- How to integrate scheduling analysis in the process?
  - When to apply it? What about tools?

It is the role of an ADL to hide those details
AADL to the rescue?

- AADL helps modeling a full system, including hardware, task sets, connections, RTOS features, ...

- All of these elements are mandatory to apply real-time scheduling theory
  - Example: an AADL model can include periodic tasks and usual scheduling policies
    - Worst case execution time (or WCET), period, deadline
    - Fixed priority scheduling

- However, in many cases, the models stay too complex
  - Dependent tasks, shared buffers or buses, ...
Summary

1. Issues about real-time scheduling: AADL to the rescue
2. Focus on fixed-priority scheduling:
   - Basics on uniprocessor
3. AADL components/properties to scheduling analysis
4. An example with Cheddar
Real-time scheduling theory: models of task

- **Task**: sequence of statements + data + state.

- **Usual task types**:
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical functions).
  - Aperiodic tasks (non-critical functions).
Real-time scheduling theory: models of task

- **Usual parameters of a periodic task i:**
  - **Period**: $P_i$ (duration between two periodic release times). A task starts a job for each release time.
  - **Deadline to meet**: $D_i$, timing constraint to meet, relative to the period/job.
  - **First task release time (first job)**: $S_i$.
  - **Worst case execution time of each job**: $C_i$ (or capacity or WCET).
  - **Priority**: allows the scheduler to choose the task to run.
Real-time scheduling theory: models of task

- Assumptions for the next slides (synchronous periodic task with deadlines on requests):
  - All tasks are periodic.
  - All tasks are independent.
  - $\forall i : P_i = D_i :$ a task must end its current job before its next release time.
  - $\forall i : S_i = 0$ => called critical instant (worst case on processor demand).
Uniprocessor usual real-time scheduling policies

- **On-line/off-line scheduling**: the scheduling is computed before or at execution time?
- **Fixed/dynamic priority scheduler**: priorities may change at execution time?
- **Preemptive or non preemptive scheduling**: can we stop a task during its execution?

- **Online, preemptive, fixed priority scheduler** with Rate Monotonic priority assignment (RM, RMS, RMA).
Uniprocessor fixed priority scheduling

- **Fixed priority scheduling**:  
  - Scheduling based on fixed priority => critical applications.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.

- **Rate Monotonic priority assignment**:  
  - Optimal assignment in the case of fixed priority scheduling and uniprocessor.
  - Periodic tasks only.
Uniprocessor fixed priority scheduling

- **Two steps:**
  1. **Rate monotonic priority assignment:**
     - the highest priority tasks have the smallest periods. Priorities are assigned off-line (e.g. at design time, before execution).
  2. **Fixed priority scheduling:**
     - at any time, run the ready task which has the highest priority level.
Uniprocessor fixed priority scheduling

- Rate Monotonic assignment and preemptive fixed priority scheduling:

  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1
Uniprocessor fixed priority scheduling

- Feasibility/Schedulability tests:
  1. **Run simulations on hyperperiod** $= [0, \text{LCM}(P_i)]$.
     Sufficient and necessary (exact result). Any priority assignment and preemptive/non preemptive scheduling.
  2. **Processor utilization factor test**:
     $$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n \cdot (2^n - 1)$$
     Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary. Does not compute an exact result.
  3. **Task worst case response time, noted $r_i$**: delay between task release time and task end time. Sometimes an exact result. Any priority assignment but preemptive scheduling.
Uniprocessor fixed priority scheduling

- **Compute** $r_i$, task $i$ worst case response time:
  - Assumptions: preemptive scheduling, synchronous periodic tasks.
  - Task $i$ response time = task $i$ capacity + delay the task $i$ has to wait for higher priority task $j$. Or:

$$R_i = C_i + \sum_{j \in hp(i)} \text{waiting time due to } j$$

or

$$R_i = C_i + \sum_{j \in hp(i)} \left\lfloor \frac{R_i}{P_j} \right\rfloor \cdot C_j$$

- $hp(i)$ is the set of tasks which have a higher priority than task $i$. $\lfloor x \rfloor$ returns the smallest integer not smaller than $x$. 
Uniprocessor fixed priority scheduling

- To compute task response time: compute $w_i^k$ with:

  $$w_i^n = C_i + \sum_{j \in h_p(i)} [w_i^{n-1}/P_j]. C_j$$

- Start with $w_i^0 = C_i$.

- Compute $w_i^1$, $w_i^2$, $w_i^3$, ... $w_i^k$ upto:
  - If $w_i^k > P_i$. No task response time can be computed for task i. Deadlines will be missed!
  - If $w_i^k = w_i^{k-1}$. $w_i^k$ is the task i response time. Deadlines will be met.
Uniprocessor fixed priority scheduling

- **Example:** T1 (P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

\[
w_1^0 = C_1 = 3 \Rightarrow r_1 = 3
\]
\[
w_2^0 = C_2 = 2
\]
\[
w_2^1 = C_2 + \left\lfloor \frac{w_2^0}{P_1} \right\rfloor . C_1 = 2 + \left\lfloor \frac{2}{7} \right\rfloor . 3 = 5
\]
\[
w_2^2 = C_2 + \left\lfloor \frac{w_2^1}{P_1} \right\rfloor . C_1 = 2 + \left\lfloor \frac{5}{7} \right\rfloor . 3 = 5 \Rightarrow r_2 = 5
\]
\[
w_3^0 = C_3 = 5
\]
\[
w_3^1 = C_3 + \left\lfloor \frac{w_3^0}{P_1} \right\rfloor . C_1 + \left\lfloor \frac{w_3^0}{P_2} \right\rfloor . C_2 = 10
\]
\[
w_3^2 = C_3 + \left\lfloor \frac{w_3^1}{P_1} \right\rfloor . C_1 + \left\lfloor \frac{w_3^1}{P_2} \right\rfloor . C_2 = 13
\]
\[
w_3^3 = C_3 + \left\lfloor \frac{w_3^2}{P_1} \right\rfloor . C_1 + \left\lfloor \frac{w_3^2}{P_2} \right\rfloor . C_2 = 15
\]
\[
w_3^4 = C_3 + \left\lfloor \frac{w_3^3}{P_1} \right\rfloor . C_1 + \left\lfloor \frac{w_3^3}{P_2} \right\rfloor . C_2 = 18
\]
\[
w_3^5 = C_3 + \left\lfloor \frac{w_3^4}{P_1} \right\rfloor . C_1 + \left\lfloor \frac{w_3^4}{P_2} \right\rfloor . C_2 = 18 \Rightarrow r_3 = 18
Example with the AADL case study:
- “display_panel” thread which displays data. P=100, C=20.
- “receiver” thread which sends data. P=250, C=50.
- “analyser” thread which analyzes data. P=500, C=150.

Processor utilization factor test:
- $U = \frac{20}{100} + \frac{150}{500} + \frac{50}{250} = 0.7$
- $\text{Bound} = 3 \left( 2^3 - 1 \right) = 0.779$
- $U \leq \text{Bound} \Rightarrow$ deadlines will be met.

Task response time: $R_{\text{analyser}} = 330$, $R_{\text{display_panel}} = 20$, $R_{\text{receiver}} = 70$.

Run simulations on hyperperiod: $[0, \text{LCM}(P_i)] = [0, 500]$. 
Uniprocessor fixed priority scheduling

Response times:
- Display panel: 20
- Receiver: 70, 250, 300 (50)
- Analyzer: 330 (330)
Fixed priority and shared resources

- Previous tasks were independent … does not really exist in true life.

- **Task dependencies**:
  - Shared resources.
    - E.g. with AADL: threads may wait for AADL protected data component access.
  - Precedencies between tasks.
    - E.g. with AADL: threads exchange data by data port connections.
Fixed priority and shared resources

- Shared resources are usually modeled by semaphores.
- **We use specific semaphores implementing inheritance protocols:**
  - To take care of priority inversion.
  - To compute worst case task blocking time for the access to a shared resource. Blocking time $B_i$.

- **Inheritance protocols:**
  - PIP (Priority inheritance protocol), can not be used with more than one shared resource due to deadlock.
  - PCP (Priority Ceiling Protocol), implemented in most of real-time operating systems (e.g. VxWorks).
  - Several implementations of PCP exists: OPCP, ICPP, …
Fixed priority and shared resources

- **What is Priority inversion:** a low priority task blocks a high priority task

  ![Diagram showing priority inversion](image)

  - \( B_i \) = worst case on the shared resource waiting time.
Fixed priority and shared resources

- **ICPP (Immediate Ceiling Priority Protocol):**
  - Ceiling priority of a resource = maximum static priority of the tasks which use it.
  - Dynamic task priority = maximum of its own static priority and the ceiling priorities of any resources it has locked.
  - Bi=longest critical section; prevent deadlocks
Fixed priority and shared resources

- How to take into account the waiting time $B_i$:
  
  - Processor utilization factor test:
    \[
    \forall \ i, \ 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{C_k}{P_k} + \frac{C_i + B_i}{P_i} \leq i \cdot \left(2^i - 1\right)
    \]
  
  - Worst case response time:
    \[
    R_i = B_i + C_i + \sum_{j \in h_p(i)} \left[\frac{R_i}{P_j}\right] \cdot C_j
    \]
To conclude on scheduling analysis

- **Many feasibility tests:** depending on task, processor, scheduler, shared resource parameters or dependencies. What about uniprocessor or multiprocessor or hierarchical or distributed?

\[ R_i = C_i + \sum_{j \in hp(i)} \left( \frac{R_i}{P_j} \right) \cdot C_j \]

- **Many assumptions:** require preemptive and fixed priority scheduling, synchronous periodic independent tasks with deadlines on requests …

Many feasibility tests …. Many assumptions …
How to choose them?
Summary

1. Issues about real-time scheduling: AADL to the rescue
2. Focus on fixed-priority scheduling:
   - Basics on uniprocessor
3. AADL components/properties to scheduling analysis
4. An example with Cheddar
AADL to the rescue?

- **Issues:**
  - Ensure all required model elements are given for the analysis
  - Ensure model elements are compliant with analysis requirements

- **AADL helps because:**
  - AADL as a pivot language between tools. International standard.
  - Close to the real-time scheduling theory: real-time scheduling concepts can be found. Ex:
    - Component categories: thread, data, processor
    - Property sets: Thread_Properties, Timing_Properties, Communication_Properties, AADL_Project
Property sets for scheduling analysis

- Properties related to processor:

  Preemptive_Scheduler : aadlboolean applies to (processor);

  Scheduling Protocol:
  inherit list of Supported_Scheduling_Protocols
  applies to (virtual processor, processor);
  -- RATE_MONOTONIC_PROTOCOL,
  -- POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL, ..
Property sets for scheduling analysis

Properties related to the threads/data:

- **Compute Execution Time**: Time Range
  applies to (thread, subprogram, ...);

- **Deadline**: inherit Time => Period applies to (thread, ...),

- **Period**: inherit Time applies to (thread, ...);

- **Dispatch Protocol**: Supported Dispatch Protocols
  applies to (thread);
  -- Periodic, Sporadic, Timed, Hybrid, Aperiodic, Backg.
  ...

- **Priority**: inherit aadlinteger applies to (thread, ..., dat)

- **Concurrency Control Protocol**: Supported Concurrency Control Protocols
  applies to (dat)
  -- None, PCP, ICPP, ...
Property sets for scheduling analysis

Example:

thread implementation receiver.impl
properties
  Dispatch_Protocol => Periodic;
  Compute_Execution_Time => 31 ms .. 50 ms;
  Deadline => 250 ms;
  Period => 250 ms;
end receiver.impl;

data implementation target_position.impl
properties
  Concurrency_Control_Protocol => PRIORITY_CEILING_PROTOCOL;
end target_position.impl;

process implementation processing.others
subcomponents
  receiver : thread receiver.impl;
  analyzer : thread analyzer.impl;
  target : data target_position.impl;
  ...
end leon2;

processor implementation leon2
properties
  Scheduling_Protocol => RATE_MONOTONIC_PROTOCOL;
  Preemptive_Scheduler => true;
end leon2;

system implementation radar.simple
subcomponents
  main : process processing.others;
  cpu : processor leon2;
  ...

Summary

1. Issues about real-time scheduling: AADL to the rescue
2. Focus on fixed-priority scheduling:
   - Basics on uniprocessor
3. AADL components/properties to scheduling analysis
4. An example with Cheddar
Cheddar: a framework to access schedulability

- **Cheddar tool =**
  - analysis framework (queueing system theory & real-time scheduling theory)
  - + internal ADL (architecture description language)
  - + various standard ADL parsers (AADL, MARTE UML)
  - + simple model editor.

- **Two versions:**
  - Open source (Cheddar): educational and research.
  - Industrial (AADLInspector): Ellidiss Tech product.

- **Supports:** Ellidiss Tech., Conseil régional de Bretagne, BMO, EGIDE/Campus France, Thales Communication

- **AADL is a rich language:** Cheddar proposes design patterns to help engineers to select relevant feasibility tests
A “design pattern” approach to increase real-time scheduling usability

- Define a set of AADL design patterns of real-time systems.
  - models a typical thread communication or synchronization.
  - set of constraints on entities of the AADL model.

- For each design pattern, define feasibility tests that can be applied according to their applicability assumptions.

- Schedulability analysis of a AADL model:
  1. Checks compliancy of the AADL model with one of the design-patterns ... which then gives which feasibility tests can be applied.
  2. Compute these feasibility tests.
A “design pattern” approach to increase real-time scheduling usability

- Specification of various design patterns:
  - **Time-triggered**: time triggered architecture (data port connection)
  - **Ravenscar**: shared data and PCP (data component).
  - **Black board**: readers/writers synchronization
  - **Queued buffer**: producer/consumer synchronization
  - ...
  - Compositions of design patterns.

- Example of the Ravenscar design-pattern.
The «Ravenscar» design pattern

- **Ravenscar:**
  - Part of the Ada 1995 standard
  - A set of guidelines/constraints to enable efficient and deterministic task scheduling of Ada programs
  - Later extended to Java RTSJ, C/POSIX, and AADL

- **Objective:** remove all that prevent Ada programs analysis
  1. All Ada tasks are either periodic or sporadic
  2. Communication through shared data, no Ada rendez-vous
  3. Shared data protected by PCP
  4. Static, no dynamic creation of Ada tasks
  5. Fixed priority preemptive scheduling similar to POSIX 1003

- **Feasibility test to compute:** worst case thread response time + thread blocking time due to data component access.
The «Ravenscar» design pattern

- Radar Example:

  thread implementation receiver.impl
    properties
      Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 31 ms .. 50 ms;
      Deadline => 250 ms;
      Period => 250 ms;
    end receiver.impl;

  data implementation target_position.impl
    properties
      Concurrency-Control-Protocol => PRIORITY_CEILING.PROTOCOL;
    end target_position.impl;

  process implementation processing.others
    subcomponents
      receiver : thread receiver.impl;
      analyzer : thread analyzer.impl;
      target : data target_position.impl;
    ...

  processor implementation leon2
    properties
      Scheduling-Protocol => RATE_MONOTONIC_PROTOCOL;
      Preemptive_Scheduler => true;
    end leon2;

  system implementation radar.simple
    subcomponents
      main : process processing.others;
      cpu : processor leon2;
    ...

The «Ravenscar» design pattern

- **Demos:**
  - Scheduling analysis of the radar example with Cheddar
  - And with AADLInspector also