AADL: about scheduling analysis
Embedded real-time critical systems have temporal constraints to meet (e.g. deadline).

Many systems are built with operating systems providing multitasking facilities ... Tasks may have deadline.

But, tasks make temporal constraints analysis difficult to do:
- We must take interference delaying a task into account: other tasks, shared resources, ...
- Need to take scheduling into account.
- Scheduling (or schedulability) analysis.
Real-Time scheduling theory

1. **A set of simplified tasks models** (to model functions of the system)

2. **A set of analytical methods** (called feasibility tests)
   - Example:
   \[
   R_i \leq \text{Deadline} \quad R_i = C_i + \sum_{j \in hp(i)} \left[ \frac{R_i}{P_j} \right] \cdot C_j
   \]

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

- Real-Time scheduling theory (uniprocessor)
  - Theoretical results defined from 1974 to 1994: feasibility tests exist for uniprocessor architectures
- Supported at a decent level since POSIX 1003 real-time operating systems and ARINC653, ...
- Industry demanding
  - Yet, hard to use
Summary

1. Issues about real-time scheduling analysis: AADL to the rescue
2. Basics on scheduling analysis: fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis
What to model to achieve early scheduling analysis

1. **Software side:**
   - Workload: release time, execution time
   - Timing constraints
   - Software entity interferences, examples:
     - Tasks relationships/communication or synchronization: e.g. shared data, data flow
     - Task containers: ARINC 653 partition, process

2. **Hardware (should be called execution platform) side:**
   - Available resources, e.g. computing capabilities
   - Contention, interference, examples: processing units, cache, memory bus, NoC, …

3. **Deployment**

=> **Architecture models**

=> It is the role of an ADL to model those elements
Real-Time scheduling theory is hard to apply

- Requires strong theoretical knowledge/skills
  - Numerous theoretical results: how to choose the right one?
  - Numerous assumptions for each result.
  - How to abstract/model a system to verify deadlines?
- How to integrate scheduling analysis in the engineering process?
  - When to apply it? What about tools?

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

- **Why AADL helps:**
  - All required model elements are given for the analysis
    - Component categories: thread, data, processor
    - Feature categories: data access, data port, ...
    - Properties: Deadline, Priority, WCET, Ceiling Priority, ...
    - Annexes (e.g. behavior annex)
  - **AADL semantic:** formal and natural language
    - E.g. automata to define the concept of periodic thread
    - Close to the real-time scheduling analysis methods
  - **Model engineering:** reusability, several levels of abstraction
  - **Tools & chain tools:** AADL as a pivot language (international standard)
    - VERSA, OSATE, POLA/FIACRE/TINA, CARTS, MAST, Marzhin, Cheddar, ... by Ocarina, TASTE, AADLInspector, RAMSES, MOSART, OSATE ...
AADL to the rescue?

- **But AADL does not solve everything:**
  - AADL is a complex language
  - How to ensure model elements are compliant with analysis requirements/assumptions, **sustainability**, accuracy, …
  - Not a unique AADL model for a given system to model
  - Not a unique mapping between a design model and an analysis model
  - Having AADL scheduling analysis tools is not enough too, how to use them?
  - …
Summary

1. Issues about real-time scheduling analysis: AADL to the rescue
2. Basics on scheduling analysis: fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis
Real-time scheduling theory: models of task

- **Task simplified model:** sequence of statements + data.

- **Usual kind of tasks:**
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical functions): have several jobs and release times.
  - Aperiodic tasks (non-critical functions): only one job and one release time.
Real-time scheduling theory: models of task

- **Usual parameters of a periodic task i:**
  - **Period:** $P_i$ (duration between two release times). A task starts a job for each release time.
  - **Deadline to meet:** $D_i$, timing constraint to meet.
  - **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.
- ∀i : Pi=Di : a task must end its current job before its next release time.
- ∀i : Si=0 => called critical instant (worst case on processor demand).
Uniprocessor fixed priority scheduling

- **Fixed priority scheduling:**
  - Scheduling based on fixed priority => priorities do not change during execution time.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.

- **Priority assignments:**
  - Rate Monotonic, Deadline Monotonic, OPA, …
Uniprocessor fixed priority scheduling

- **Rate Monotonic:**
  - Optimal priority assignment in the case of fixed priority scheduling and uniprocessor.
  - Periodic tasks.
  - The highest priority tasks have the smallest periods.
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

  Deadline of T1
  Deadline of T1
  Deadline of T2
  T2 is preempted
Uniprocessor fixed priority scheduling

- Feasibility/Schedulability tests to predict on design-time if deadline will be met:
  1. Run simulations on feasibility interval $= [0, \text{LCM}(P_i)]$. Sufficient and necessary condition.
  2. Processor utilization factor test:
     $$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n \cdot \left(\frac{1}{2^n} - 1\right)$$ (about 69%)
     Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary condition.
  3. Task worst case response time, noted $R_i$ : delay between task release time and task completion time. Any priority assignment, preemptive scheduling.
Uniprocessor fixed priority scheduling

- **Compute Ri, task i worst case response time:**
  - Task i response time = task i capacity + delay the task i has to wait for higher priority task j. Or:
  
  \[
  R_i = C_i + \sum_{j \in hp(i)} \text{waiting time due to } j \\
  \text{or } R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j
  \]
  
  - \(hp(i)\) is the set of tasks which have a higher priority than task i.
  - \([x]\) returns the smallest integer not smaller than x.
Uniprocessor fixed priority scheduling

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

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

- 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)

\[
\begin{align*}
w_1^0 &= C_1 = 3 \Rightarrow R_1 = 3 \\
w_2^0 &= C_2 = 2 \\
w_2^1 &= C_2 + \frac{w_2^0}{P_1} \cdot C_1 = 2 + \frac{2}{7} \cdot 3 = 5 \\
w_2^2 &= C_2 + \frac{w_2^1}{P_1} \cdot C_1 = 2 + \frac{5}{7} \cdot 3 = 5 \Rightarrow R_2 = 5 \\
w_3^0 &= C_3 = 5 \\
w_3^1 &= C_3 + \frac{w_3^0}{P_1} \cdot C_1 + \frac{w_3^1}{P_2} \cdot C_2 = 10 \\
w_3^2 &= C_3 + \frac{w_3^1}{P_1} \cdot C_1 + \frac{w_3^1}{P_2} \cdot C_2 = 13 \\
w_3^3 &= C_3 + \frac{w_3^2}{P_1} \cdot C_1 + \frac{w_3^2}{P_2} \cdot C_2 = 15 \\
w_3^4 &= C_3 + \frac{w_3^3}{P_1} \cdot C_1 + \frac{w_3^3}{P_2} \cdot C_2 = 18 \\
w_3^5 &= C_3 + \frac{w_3^4}{P_1} \cdot C_1 + \frac{w_3^4}{P_2} \cdot C_2 = 18 \Rightarrow R_3 = 18
\end{align*}
\]
Uniprocessor fixed priority scheduling

- **Example with the AADL radar 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:**
  - \(U = \frac{20}{100} + \frac{150}{500} + \frac{50}{250} = 0.7\)
  - \(\text{Bound} = 3 \cdot (2^\frac{1}{3} - 1) = 0.779\)
  - \(U \leq \text{Bound} \Rightarrow \text{deadlines will be met.}\)

- **Worst case task response time:** \(R_{\text{analyser}} = 330,\)
  \(R_{\text{display panel}} = 20, R_{\text{receiver}} = 70.\)

- **Run simulations on feasibility interval:** \([0, \text{LCM}(P_i)] = [0, 500].\)
Uniprocessor fixed priority scheduling

Response times = 20

Display Panel

Response time = 70

Receiver

Response time = 330

Analyzer

Response time = 50

Response time = 300
Fixed priority and shared resources

- Previous tasks were independent … does not exist in real 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 can be modeled by semaphores for scheduling analysis.

- **We use specific semaphores implementing inheritance protocols:**
  - To take care of priority inversion.
  - To compute worst case task waiting time for the access to a shared resource => Blocking time $B_i$.

- **Inheritance protocols:**
  - PIP (Priority inheritance protocol), cannot 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 task blocking and preemption]

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

- **PIP (Priority Inheritance Protocol):**
  - A task which blocks a higher priority task runs its critical section with the priority level of the blocked task.
  - Only one shared resource, deadlock otherwise.
  - \( B_i = \text{sum of critical section durations of lower priority tasks than i} \)
Fixed priority and shared resources

ICPP (Immediate Ceiling Priority Protocol):

- Ceiling priority of a resource = maximum fixed priority of the tasks which use it.
- Dynamic task priority = maximum of its own fixed priority and the ceiling priorities of any resources it has locked.
- $B_i$ = longest critical section; prevent deadlock and reduce blocking
Fixed priority and shared resources

- **How to take into account Bi (blocking time):**

  - Processor utilization factor test:
    \[ \forall i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{Ck}{Pk} + \frac{Ci+Bi}{Pi} \leq i \cdot (2^i - 1) \]

  - Worst case response time:
    \[ R_i = B_i + C_i + \sum_{j \in hp(i)} \left[ \frac{R_i}{P_j} \right] \cdot C_j \]
To conclude on scheduling analysis

- **Many feasibility tests**: depending on task, processor, scheduler, shared resource, dependencies, multiprocessor, hierarchical, distributed…

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

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

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

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

\[
R_i = w_i + J_i
\]

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

Many feasibility tests... Many assumptions…

How to choose them?
Summary

1. Issues about real-time scheduling analysis: AADL to the rescue
2. Basics on scheduling analysis: fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis
AADL to the rescue?

- **Issues:**
  1. Ensure all required model elements are given for the analysis
  2. Ensure model elements are compliant with analysis requirements/assumptions

- **AADL helps for the first issue:**
  - AADL as a pivot language between tools. International standard.
  - Close to the real-time scheduling theory: real-time scheduling analysis concepts can be found. Ex:
    - Component categories: thread, data, processor
    - **Property:** Deadline, Fixed Priority, ICPP, Ceiling Priority, ...
Property sets for scheduling analysis

- **Properties related to processor components:**

  - **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 components:**

  **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, Background, ...

  **Priority**: inherit aadlinteger applies to (thread, …, data);

  **Concurrency Control Protocol**: Supported Concurrency Control Protocols applies to (data);
  -- None, PCP, ICPP, …
AADL to the rescue?

- **Issues:**
  1. Ensure all required model elements are given for the analysis
  2. Ensure model elements are compliant with analysis requirements/assumptions

- **And for the second issue?**
Cheddar : a framework to assess schedulability of AADL models

- **Cheddar tool** =
  + analysis framework (queueing system theory & real-time scheduling theory)
  + internal ADL (analysis model)
  + simple analysis model editor
  + optimization tools
  + ...

- **Two versions:**
  - Open source (Cheddar) : teaching/research, TASTE, OSATE, MOSART, RAMSES, ...
  - Commercial product (AADLInspector) : Ellidiss Tech product.

- **Supports** : Ellidiss Tech., Conseil régional de Bretagne, Brest Métropole, Campus France, BPI France
Cheddar: main analysis features
(see http://beru.univ-brest.fr/~singhoff/cheddar)

- **Analysis by scheduling simulations:**
  - Various scheduling policies, uniprocessor, multiprocessor, cache, …
  - Simulation data analysis
- **Task schedulability/feasibility tests**
- **Design space exploration methods**
- **Task and resource priority assignments**
- **Partitioning algorithms**
- **Queueing system theory models/buffer feasibility tests**
- **Modeling/analysis with task dependencies**
AADL “design pattern” approach to automatically perform scheduling analysis

- Let assume we have to evaluate a given architecture model in a design exploration flow.

- **Problem statement reminder:**
  - Numerous schedulability tests; how to choose the right one?
  - Numerous assumptions for each schedulability test; how to enforce them for a given model?
  - How to automatically perform scheduling analysis?
Approach:

- Define a set of AADL architectural design patterns of real-time (critical) systems:
  - models a typical thread communication or synchronization + a typical execution platform
  - set of constraints on entities/properties of the model.

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

- Schedulability analysis of an AADL model:
  1. Check compliancy of his model with one of the design-patterns … which then gives him which schedulability tests we can apply.
  2. Perform schedulability verification.
Design pattern compliancy verification

- **Top right part**: real-time system architecture model to verify.
- **Bottom right part**: modeling of a feasibility test applicability assumption.
- **Left part**: result of the model compliancy analysis.
Example: «Ravenscar» design pattern

- Specification of various design patterns:
  - **Time-triggered**: sampling data port communication between threads
  - **Ravenscar**: PCP shared data communication between threads
  - **Queued buffer/ARINC653**: producer/consumer synchronization
  - **Black board/ARINC653**: readers/writers synchronization
  - ...
  - Compositions of design patterns.

- **Ravenscar**: used by TASTE/ESA

- **Constraints defining “Ravenscar” to perform the analysis with a given schedulability test:**
  - Constraint 1: all threads are periodic
  - Constraint 2: threads start at the same time
  - Constraint 3: shared data with PCP
  - Constraint n: fixed preemptive priority scheduling + uniprocessor
  - ...

Constraints defining “Ravenscar” to perform the analysis with a given schedulability test:
Example: «Ravenscar» compliant AADL model

**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;

...
Demos, practical labs

- Scheduling analysis of the radar example with AADLInspector & OSATE/Cheddar