About Early Scheduling Verification Of Embedded Real-Time Critical Systems: An Example With AADL

F. Singhoff+, S. Rubini+, N. Tran+, M. Dridi+,
J. Boukhobza+, A. Plantec+,
L. Lemarchand+, J.P. Diguet*, P. Dissaux*, J. Legrand*,
A. Schach*, V.A. Nicolas+

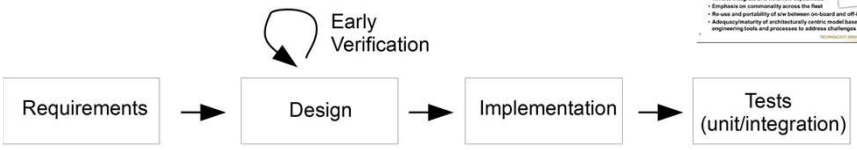
+ Lab-STICC UMR CNRS 6285/UBO
*** Ellidiss Technologies**


Summary

We focus on the design of Real-Time, Critical, Embedded Systems

- **Real-time systems:**
  - Systems that have temporal constraints which must be met, e.g. functions must complete computation before or on deadlines.
- **Critical systems:**
  - Defects (e.g. missed deadlines) could have a dramatic impact on human life, on the system, …
- **Embedded systems:** computing system designed for specific control functions within a larger system.
  - Part of a complete device, including hardware and mechanical parts most of the time
  - Limited amount of resources: energy, memory, computation capabilities, …
- **Examples:** aircraft, satellite, automotive, …

About scheduling analysis and early verification



- **Motivation for early verification (NIST 2002):**
  - 70% of fault are introduced during the design step ; Only 3% are found/solved. Cost : x1
  - Integration test step and latter: 80% bugs are found/solved. Cost : x20
- **Objective:** increase the number of faults found at design step!
- **Multiple early verifications**, including expected performances, i.e deadlines of functions can be met?

Problem statement

- **Issues:**
  - How to model/design a real-time critical embedded system?
  - How to early verify the solution?
  - How to (automatically if possible) prototype/implement it?
- **One solution among others: use an ADL, i.e. architecture description language.**

Problem statement

- How to model for scheduling analysis with AADL (*Architecture Analysis and Design Language*), an ADL for real-time embedded critical systems?
- Focus on concurrent applications, composed of several tasks.
- Focus on deadline verifications: real-time scheduling analysis theory

Summary

1. **Problem statement**
2. **AADL, an architecture description language for embedded critical real-time systems**
3. **Introduction on real-time scheduling analysis**
4. **Research trends about real-time scheduling analysis**
5. **Conclusion**

7/41

AADL is for Analysis

- **ADL, Architecture Description Language:**
  - **Goal:** modeling software and hardware architectures
  - **Concepts:** components, connections, deployments.
- International standard promoted by SAE, AS-2C committee, released as AS5506 family of standards
- **Core language document:** AADL 2.2 (AS 5506C) 2017
- **Annex documents to address specific concerns:** ARINC653, Data Modelling, Code Generation, Behavior, Error Model

8/41

AADL components

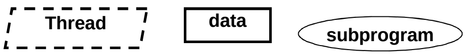
- **AADL model**: hierarchy/tree of components
  - Composition hierarchy (subcomponents)
  - Inheritance hierarchy (extends)
  - …
- **AADL component:**
  - Model a software or a hardware entity
  - Has a type/interface, zero, one or several implementations
  - May have properties: similar to UML attributes
  - May be organized in packages: **reusable**

AADL components

- **How to declare an AADL component?**
  - Component type: name, category, properties, features => interface
  - Component implementation: internal structure (subcomponents), properties
- **Component categories:** model real-time abstractions, close to the implementation space (e.g. task).
  - Each category has well-defined semantics/behavior, refined through properties and annexes
  - Hardware components: execution platform
  - Software components
  - Systems : bounding box of a system. Model deployments.

Main software component categories

- **thread :** schedulable execution flow, Ada or VxWorks task, Java or POSIX thread. Execute programs
- **data :** data placeholder, e.g. C struct, C++ class, Ada record
- **process :** application. It must hold at least one thread
- **subprogram :** a sequential execution flow. Associated to a source code (C, Ada) or a model (SCADE, Simulink)



Examples of component

```
                                                -- sequential
control flow
  subprogram Spg                                --  Spg represents a C
function,
  features                                      --  in file
"foo.c", that takes one
    in_param : in parameter foo_data;           -
  properties
    Source_Language => (C);
    Source_Text => ("foo.c");
  end Spg;

                                                --
schedulable control flow
  thread bar_thread                             --  bar_thread is a
periodc thread :
  features                                      --
dispatched every 10 ms
    in_data : in data port foo_data;
  properties
    Dispatch_Protocol => Periodic;
    Period => 10 ms;
 end bar_thread;
```

Standard properties, one can define its own properties

```
thread implementation bar_thread.impl     --  in this implementation, at
                                          --
  parts
    C : { S : subprogram spg; };          --  sequence. We p
  connections                             --  paramete
    parameter in_data -> S.in_param;
end bar_thread.impl;
```

Main AADL Tools

- **OSATE** (SEI/CMU, http://aadl.info)
  - Eclipse-based tools. Reference implementation.
  - Textual and graphical editors + various analysis plug-ins
- **MASIW (**ISPRAS, http://masiw.ispras.ru)
  - Graphical editor, various analysis, configuration generation.
- **Ocarina** (ISAE, http://www.openaadl.org)
  - Command line tool, library to manipulate models.
  - AADL parser + code generation + analysis (Petri Net, WCET, …)
- **STOOD, AADLInspector** (Ellidiss, http://www.ellidiss.com)
  - Graphical editor, code/documentation generation
  - Scheduling analysis (commercial version of Cheddar)
- **Cheddar** (UBO/Lab-STICC, http://beru.univ-brest.fr/~singhoff/cheddar/)
  - Scheduling analysis
- **Many others:** RAMSES, PolyChrony, ASSIST, MDCF, TASTE, Scade Architect, Camet, Bless

Summary

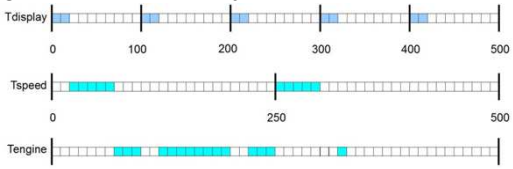
1. **Problem statement**
2. **AADL, an architecture description language for embedded critical real-time systems**
3. **Introduction on real-time scheduling analysis**
4. **Research trends about real-time scheduling analysis**
5. **Conclusion**

Real-time scheduling theory, to early verify task deadlines

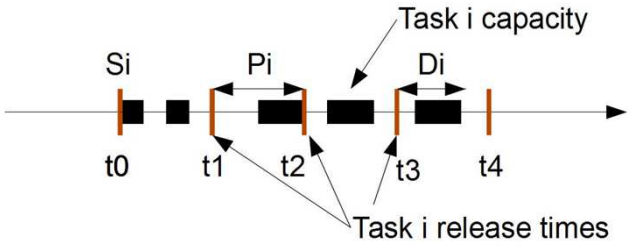
1. **A set of simplified task models (AADL is compliant with):**
   - ❑ To model functions of the system
   - ❑ E.g. Periodic task model (Lui and Layland 1974)

2. **Schedulability tests (perform deadline verifications):**
   1. **Analytical models:** equations modeling the temporal behavior of the task set (e.g. Joseph and Pandia 1986)

   2. **Scheduling simulations methods:** over the feasibility interval.
      - ❑ **Feasibility interval:** interval of time such that if all the deadlines in the interval are met, then deadlines are met during all the life of the system.



Periodic task model, Lui and Layland 1974



- ❑ **Usual parameters of a periodic task i:**
  - ❑ **Period:** Pi (fixed duration between two release times). A task starts a job for each release time.
  - ❑ **Worst case execution time of each job:** Ci (or capacity or WCET).
  - ❑ **Deadline to meet:** Di, timing constraint to meet.
  - ❑ **First task release time (first job):** Si.
  - ❑ **Priority:** allows the scheduler to choose the task to run

Periodic task model, Lui and Layland 1974

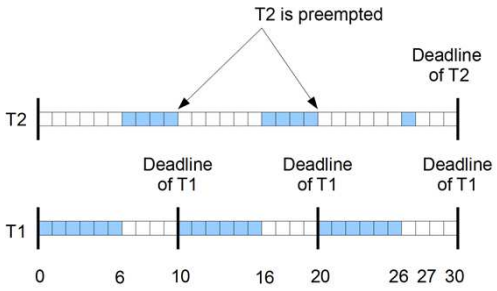
- **Assumptions for the next slides (synchronous periodic task with deadlines on requests):**
  - All tasks are periodic.
  - All tasks are independent.
  - ∀i : Pi=Di : a task must end its current job before its next release time.
  - ∀i : Si=0 => called critical instant (worst case on processor demand).

Uniprocessor preemptive 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).
  - Preemptive: the scheduler is allowed to stop a low priority task when a high priority task becomes ready
  - Efficient and simple schedulability tests.
- **Priority assignment:**
  - **Rate Monotonic:** the highest priority tasks have the smallest periods
- **Dedicated uniprocessor**

Uniprocessor preemptive fixed priority scheduling

- **Rate Monotonic assignment and preemptive fixed priority scheduling:**



- T1 : C1=6, P1=10, Prio1=high
- T2 : C2=9, P2=30, Prio2=low
- **Sustainable scheduling simulation:** deadlines met in the worst case scenario will be met during execution time.

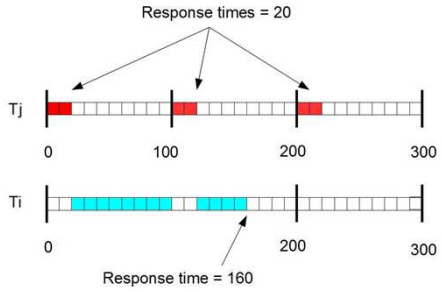
Uniprocessor preemptive fixed priority scheduling

- **Examples of schedulability tests <u>to predict on design-time if deadline will be met</u>:**

1. **Run simulations on feasibility interval** = [0,LCM(Pi)] Sufficient and necessary condition.
2. **Processor utilization test:**
   $U = \sum_{i=1}^{n} Ci/Pi \leq n.(2^{\frac{1}{n}}-1)$ (about 69%)
   Rate Monotonic assignment. Sufficient but not necessary condition.
3. **Task worst case response time test**: delay between task release time and completion time. Any priority assignment.
4. **…**

Uniprocessor preemptive fixed priority scheduling

- **Ri, task i worst case response time:**
  - Task i response time = task i worst case execution time + delay task i has to wait for higher priority task j. Or:



  - Analysis of interferences due to task j.

Uniprocessor preemptive fixed priority scheduling

- **Ri, task i worst case response time (joseph and pandia 1986):**
  - Task i response time = task i worst case execution time + delay task i has to wait for higher priority task j. Or:

$$R_i = C_i + \sum_{j \in hp(i)} waiting\ time\ due\ to\ j \qquad or\ R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

  - hp(i) is the set of tasks which have a higher priority than task i.
  - $\lceil x \rceil$ returns the smallest integer higher or equal than x.

Real-time scheduling theory is hard to apply

- Main real-time scheduling theoretical schedulability tests defined between 1974 and 1994.
- Efficient schedulability tests exist for **dedicated uniprocessor architectures.**
- Now supported at a decent level by POSIX 1003 real-time operating systems, ARINC653, …
- Yet, hard to use

23/41

Real-time scheduling theory is hard to apply

- **Many schedulability tests ...**
  Many assumptions for each test …
- Requires strong theoretical knowledge/skills
  - How to choose the right schedulability tests?
  - How to be sure a given AADL model is compliant with all schedulability tests assumptions?
  - How to abstract/model a system to verify deadlines?

24/41

Cheddar project : how to apply real-time scheduling analysis at early steps?

- Started in 2002 by U. of Brest, partnership with Ellidiss Tech. since 2008 (industrial support).
- Cheddar tool (open source, http://beru.univ-brest.fr/~singhoff/cheddar), AADLInspector (commercial product, http://www.ellidiss.fr/public/wiki/wiki/inspector)
- Modeling language: AADL as a driving line since 2004
- **Other contributors:** Télécom-Paris-Tech (L. Pautet, E. Borde), ISAE (J. Hugues), Univ. Lisboa (J. Rufino), Univ. Sfax (B. Zalila)
- **Main supports:** Ellidiss Tech., Brittany council, Brest City, Finistère council, Thalès communication, EGIDE/Campus France

25/41

AADL design patterns (Singhoff and Plantec and Dissaux 2010, Gaudel 2014)

- **Define a set of AADL architecture design patterns:**
  = models a typical thread communication or synchronization
  + a typical execution platform
  = set of constraints on entities/properties of the AADL model.
- **For each design pattern,** define schedulability tests that are allowed to be applied according to their assumptions.
- **Schedulability analysis of an AADL model:**
  1. Check compliancy of the AADL model with one of the design-patterns … which then automatically gives which schedulability tests we can apply.
  2. Run schedulability test (deadlines are met?).

26/41

AADL models correct-by-construction for scheduling analysis

- **Examples of Cheddar AADL design patterns:** Time-triggered, Ravenscar, Queued buffer/ARINC653, Black board/ARINC653, …
- **Ravenscar** (Burns 1999)**:** used by TASTE (European Space Agency design platform integrating Cheddar/AADLInspector).
- **Constraints defining "Ravenscar":**
  - 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

27/41

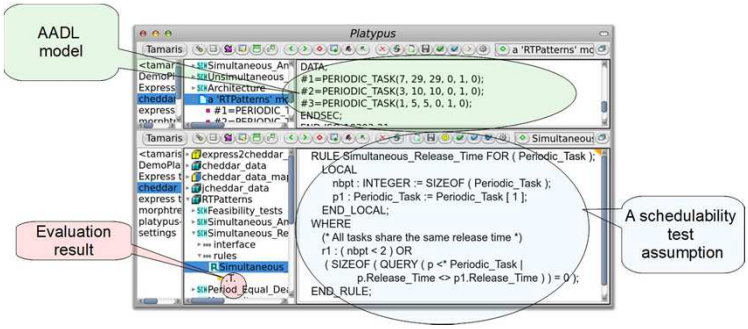
Example: an AADL model compliant with the «Ravenscar» design pattern

```
thread implementation ordo_bus.impl
  properties
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time =>  3 ms ..  7 ms;
    Deadline  =>  29 ms;
    Period =>  29 ms;
end ordo_bus.impl;

data implementation black.impl
  properties
    Concurrency_Control_Protocol
        => PRIORITY_CEILING_PROTOCOL;
end blanck.impl;
```

```
process implementation Application.impl
  subcomponents
    donnees : thread camera.impl;
    ordo_bus : thread ordo_bus.impl;
    target : data black.impl;
    . . .
processor implementation leon2
  properties
    Scheduling_Protocol =>
          RATE_MONOTONIC_PROTOCOL;
    Preemptive_Scheduler => true;
end leon2;

system implementation example
subcomponents
  process1 : process Application.impl;
  cpu1 : processor leon2;
  . . .
```

28/41

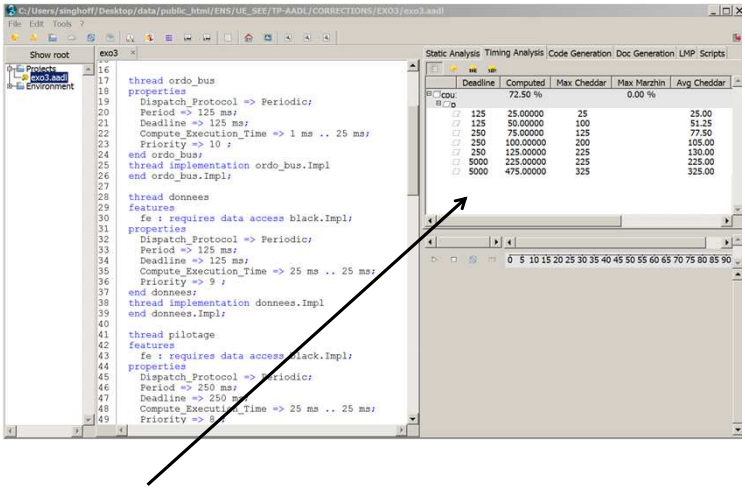
Design pattern compliancy verification



- **Top right part:** real-time system model to verify.
- **Bottom right part:** modeling of a schedulability test assumption.
- **Left part:** result of the model compliancy analysis.

29/41

«Ravenscar» scheduling analysis with Cheddar/AADLInspector



Worst case response time schedulability test

30/41

Summary

1. **Problem statement**
2. **AADL, an architecture description language for embedded critical real-time systems**
3. **Introduction on real-time scheduling analysis**
4. **Research trends about real-time scheduling analysis**
5. **Conclusion**

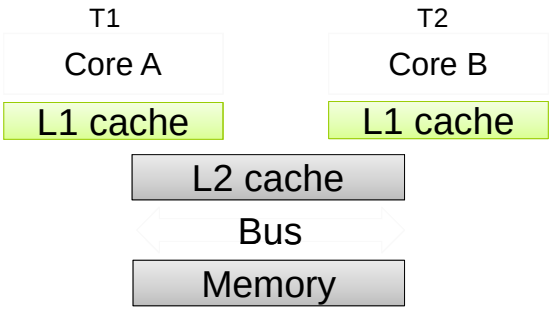
Research trends about real-time scheduling analysis (in the Lab-STICC)

- **Increased use of « Commercial Off The Shelf » processors** (Federal aviation administration, 2011)**:**
  - Multicore architectures, with shared resources such as cache units
  - Schedulability analysis with shared cache units?
- **Mix-criticality systems**
  - High amount of computation/communication capabilities shared by applications with different levels of cricitality
  - Schedulability analysis for Real-Time Networks on Chip?
- **Multi-objective partitioning:**
  - Assign tasks to processors according to safety, security and schedulability trade-off?
  - Pareto solution set, PAES meta-heuristic?

Multicore scheduling analysis

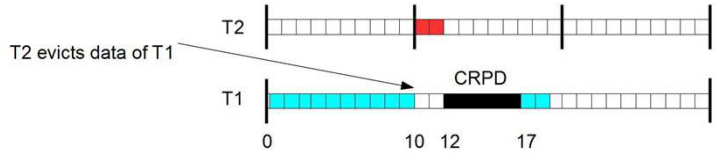
- **Specific scheduling policies:**
  1. Extension of classical uniprocessor scheduling policies such as fixed priority or EDF (but not optimal anymore)
  2. Or specific policies such as Proportional Fair (Anderson 2000), EDZL (Cho 2002), LLREF (Cho 2006), RUN (Regnier 2011), …
- **Need specific theoretical results:**
  - Schedulabilty tests
  - Feasibility interval for scheduling simulation
  - Sustainability analysis
- **Hardware shared resources create new interferences, then new delays**

Multicore scheduling analysis & shared resources



- T1 and T2 are functionally independent but are not actually independent due to the hardware resources.
- A task may be delayed due to contentions on shared hardware resources (cache, NoC, bus).

Cache memory & CRPD



- **Cache related preemption delay (CRPD): the additional time to refill the cache with memory blocks evicted by preemption.**
- **CRPD is a high preemption cost, up to 44% of the WCET of task**. (Pellizzoni 2007).
- **Variability of CRPD can be high too => make schedulability difficult**

35/41

Cache memory & scheduling analysis

- **Problem statement 1:** classical priority assignments are either not optimal or not applicable with cache memory.
- **Problem statement 2:** CRPD-aware scheduling simulation is still an open issue.
  - Scheduling simulation is non sustainable.
  - Feasibility interval of time is unknow
  - Few tools

36/41

Cache/CRPD-Aware Priority Assignment Algorithm

- Extend (Audsley 1995) priority assignment algorithm with CRPD (Tran Hai and Rubini and Boukhobza and Singhoff 2017).
- 4 solutions with different levels of pessimism, efficiency and complexity to compute an upper-bound CRPD Interference.

Solution	Description	Complexity
1. CPA-ECB	Based on Evicted cache blocs	$O(n)$
2. CPA-PT	Evaluate all **potential preemption** points.	$O(\binom{n}{n/2})$
3. CPA-PT-Simplified	More pessimistic **CRPD** computation with less complexity	$O(n \log n)$
4. CPA-Tree	Evaluate all possible preemption sequences.	$O(n!)$

37/41

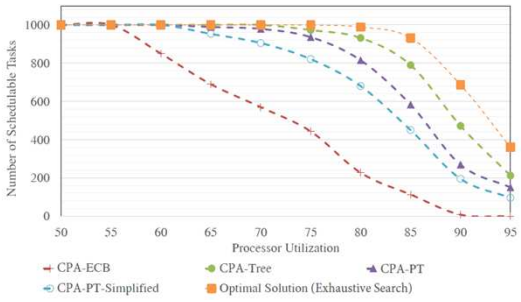
Cache/CRPD-Aware Priority Assignment Algorithm

- **Result**: **all task sets assumed to be schedulable by our priority assignment are actually schedulable over the feasibility interval.**

Weighted Schedulability Measure

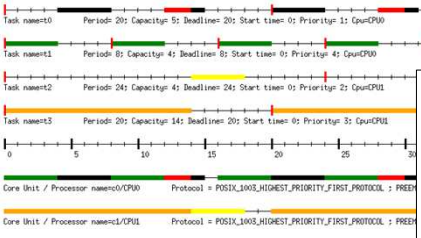
	Reuse Factor	
	0.3	0.6
CPA-ECB	42 %	41 %
CPA-PT-Simplified	65 %	50 %
CPA-PT	72 %	56 %
CPA-Tree	80 %	65 %
Exhaustive Search	87 %	74 %

Number of task set found schedulable (RF=0.3)



38/41

Cache-Aware Scheduling Simulation



Scheduling analysis for systems with cache

Cache Configuration	Control Flow Graph	Worst-Case Execution Time
Memory Layout	Cache Access Profile	Scheduling Policy

- **First step to scheduling simulation with cache** (Tran Hai and Rubini and Boukhobza and Singhoff 2017).**:**
  - Very limited: L1 uniprocessor instruction caches only
  - But sustainable CPRD model (prooved): `load block time ≤ task execution time`
  - And known feasibility interval (prooved): `[0, LCM(Pi)]`
- **Need various parameters**: cache profile, task CFG, memory layout

39/41

Summary

1. **Problem statement**
2. **AADL, an architecture description language for embedded critical real-time systems**
3. **Introduction on real-time scheduling analysis**
4. **Research trends about real-time scheduling analysis**
5. **Conclusion**

40/41

Conclusion

- **Introduction to early real-time scheduling analysis & AADL**
- **Lessons learnt and research perspectives:**
  - **Uniprocessor**: theory and tools are ready, but stays difficult to use by practitioners
  - **Open issues in scheduling analysis:** multi & manycore architectures, Mix-criticality, partitioning according to multiple objective functions, many others
  - **Papers, tools, model, evaluation data and details:** read further, http://beru.univ-brest.fr/~singhoff/cheddar
  - **Presented algorithms & approaches**: play with them with the Cheddar tool

41/41