An example of early scheduling analysis with AADL

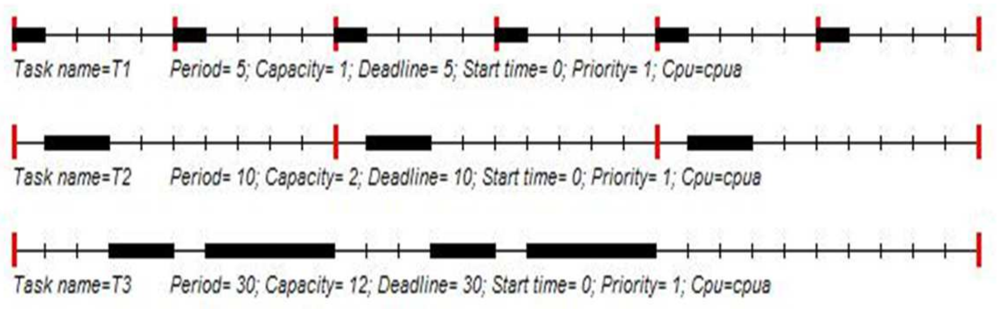
S. Rubini+, N. Tran+, M. Dridi+, V. Gaudel+, J. Boukhobza+,
A. Plantec+, C. Fotsing+, F. Singhoff+,
P. Dissaux*, J. Legrand*, A. Schach*

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

About scheduling analysis

- **Simplified models of functions :** e.g. periodic task: processor demand + deadline.
- **Analysis:** feasibility tests, simulations, formal methods, …

1. **Scheduling Simulation**:



Task name=T1 Period= 5; Capacity= 1; Deadline= 5; Start time= 0; Priority= 1; Cpu=cpua

Task name=T2 Period= 10; Capacity= 2; Deadline= 10; Start time= 0; Priority= 1; Cpu=cpua

Task name=T3 Period= 30; Capacity= 12; Deadline= 30; Start time= 0; Priority= 1; Cpu=cpua

2. **Schedulability tests** :

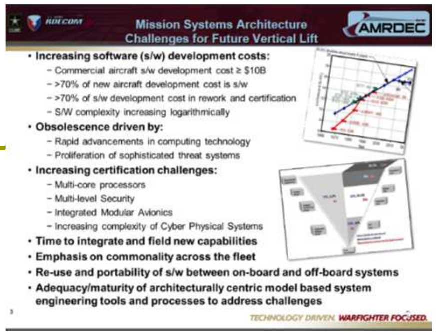
$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{P_j} \right\rceil \cdot C_j$$

3. **Formal methods (e.g. model checking)**

Not used as much we can expect ☹

Summary

About scheduling analysis and early verification



Mission Systems Architecture Challenges for Future Vertical Lift

- **Increasing software (s/w) development costs:**
  - Commercial aircraft s/w development cost ≥ \$10B
  - >70% of new aircraft development cost is s/w
  - >70% of s/w development cost in rework and certification
  - S/W complexity increasing logarithmically
- **Obsolescence driven by:**
  - Rapid advancements in computing technology
  - Proliferation of sophisticated threat systems
- **Increasing certification challenges:**
  - Multi-core processors
  - Multi-level Security
  - Integrated Modular Avionics
  - Increasing complexity of Cyber Physical Systems
- **Time to integrate and field new capabilities**
- **Emphasis on commonality across the fleet**
- **Re-use and portability of s/w between on-board and off-board systems**
- **Adequacy/maturity of architecturally centric model based system engineering tools and processes to address challenges**

TECHNOLOGY DRIVEN. WARFIGHTER FOCUSED.

- **Motivations for early verification (source AMRDEC):**
  - 70% of fault are introduced during the design step ; Only 3% are found/solved. Cost : x1
  - Unit test step: 20% of fault are introduced ; 16% are found/solved. Cost : x5
  - Integration test step: 10% of fault are introduced ; 50% are found/solved. Cost : x16

- **Objective:** increase the number of faults found at design step!
- **Early verification:** multiple verifications, including expected performances, i.e deadlines can be met?

About scheduling analysis and early verification

1. **How and what to model, in order to achieve early verifications?**

2. **Scheduling analysis requires advanced skills:**
   - ❑ Numerous theoretical results: how to choose the right one ?
   - ❑ Numerous assumptions for each result.
   - ❑ What to model ? What to abstract ?

3. **Engineers must be helped to use tools and methods:**
   - ❑ With which design languages ?
   - ❑ How to use scheduling tools ?

4. **...**

What is the role of an Architecture Description Language?

Summary

Cheddar project : context and motivations

- **Motivations : 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, educational, research), AADLInspector (commercial product)
  - 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), IUC (C. Fotsing)
  - **Main supports :** Ellidiss Tech., Brittany council, Brest City, Finistère council, Thalès communication, EGIDE/Campus France

Cheddar project : context and motivations

1. **Propose, implement (early) scheduling analysis (Cheddar), investigate how to use them with AADL (AADLInspector)**

2. **What to abstract from software and execution platform to achieve early scheduling analysis ?**
   - ❑ AADL (modeling language), Cheddar ADL (analysis language)
   - ❑ Scalability? Accuracy? Sustainability? To enforce analysis.

3. **How to automatically perform scheduling analysis?**

4. **How to achieve tools interoperability ?**
   - ❑ Various involved tools: model editors (Stood), code generators (Ocarina), WCET (aiT), system-C simulators, design exploration tools (Ramses), scheduling analysis (Cheddar), tool chains (TASTE)
   - ❑ Relationships between tools, formalize data interoperability

Summary

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 Architecture Description Language to model those elements

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/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

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:**
  - 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 ?

AADL “design pattern” approach 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.

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
  - …

Example : «Ravenscar» design pattern

- **Properties related to processor component:**

```
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, ..
```

Example : «Ravenscar» design pattern

❑ **Properties related to 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, Backg
...

 Priority: inherit aadlinteger applies to (thread, …, da

 Concurrency_Control_Protocol:
  Supported_Concurrency_Control_Protocols applies to (da
 --  None, PCP, ICPP, …
```

Example : «Ravenscar» design pattern

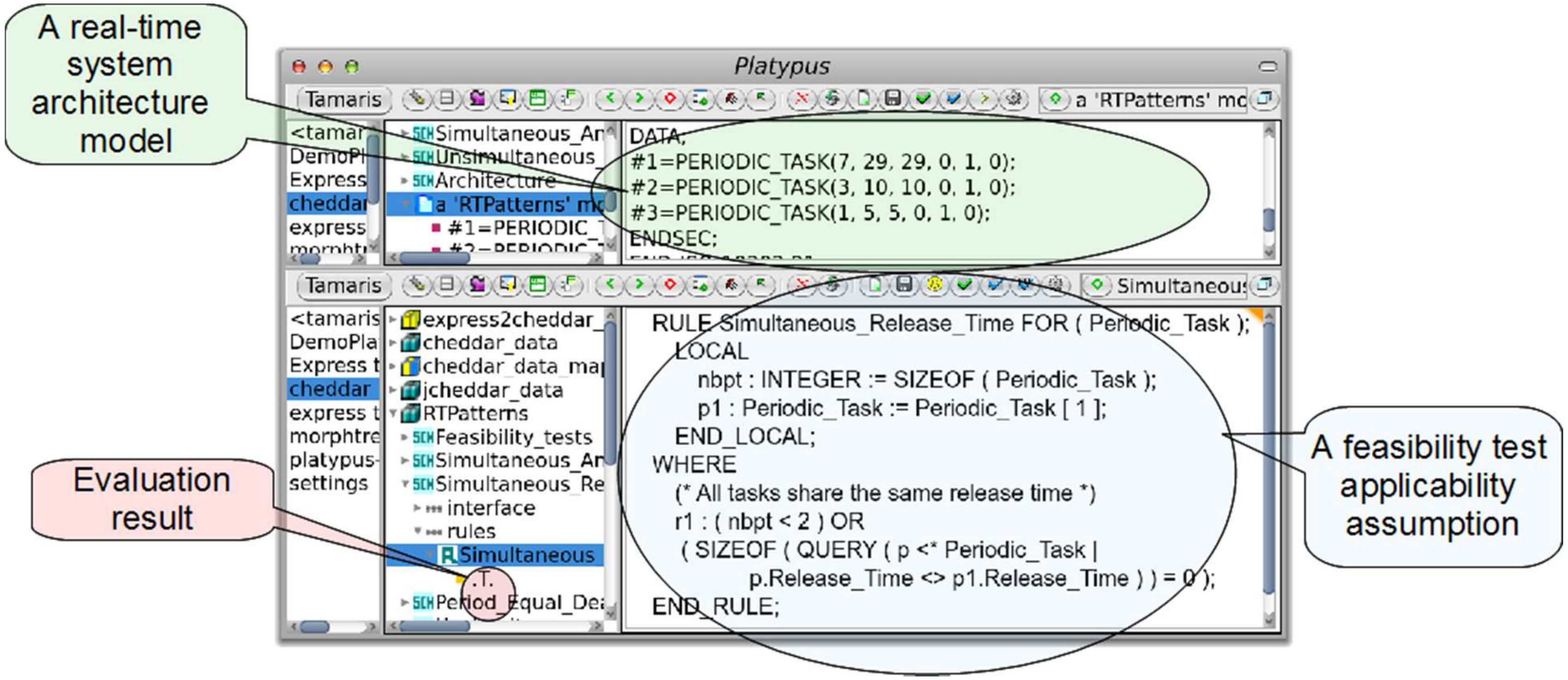
```
thread implementation ordo_bus.impl
  properties
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time =>  31 ms ..  50 ms;
    Deadline  =>  250 ms;
    Period =>  250 ms;
end ordo_bus.impl;

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

```
process implementation Application.impl
  subcomponents
    camera : thread camera.impl;
    ordo_bus : thread ordo_bus.impl;
    target : data black.impl;
    . . .
processor implementation leon2
  properties
    Scheduling_Protocol =>
            POSIX1003_HPF_PROTOCOL;
   Preemptive_Scheduler => true;
end leon2;

system implementation finder
subcomponents
  process1 : process soft.impl;
  cpu1 : processor leon2;
   . . .
```

Design pattern compliancy verification

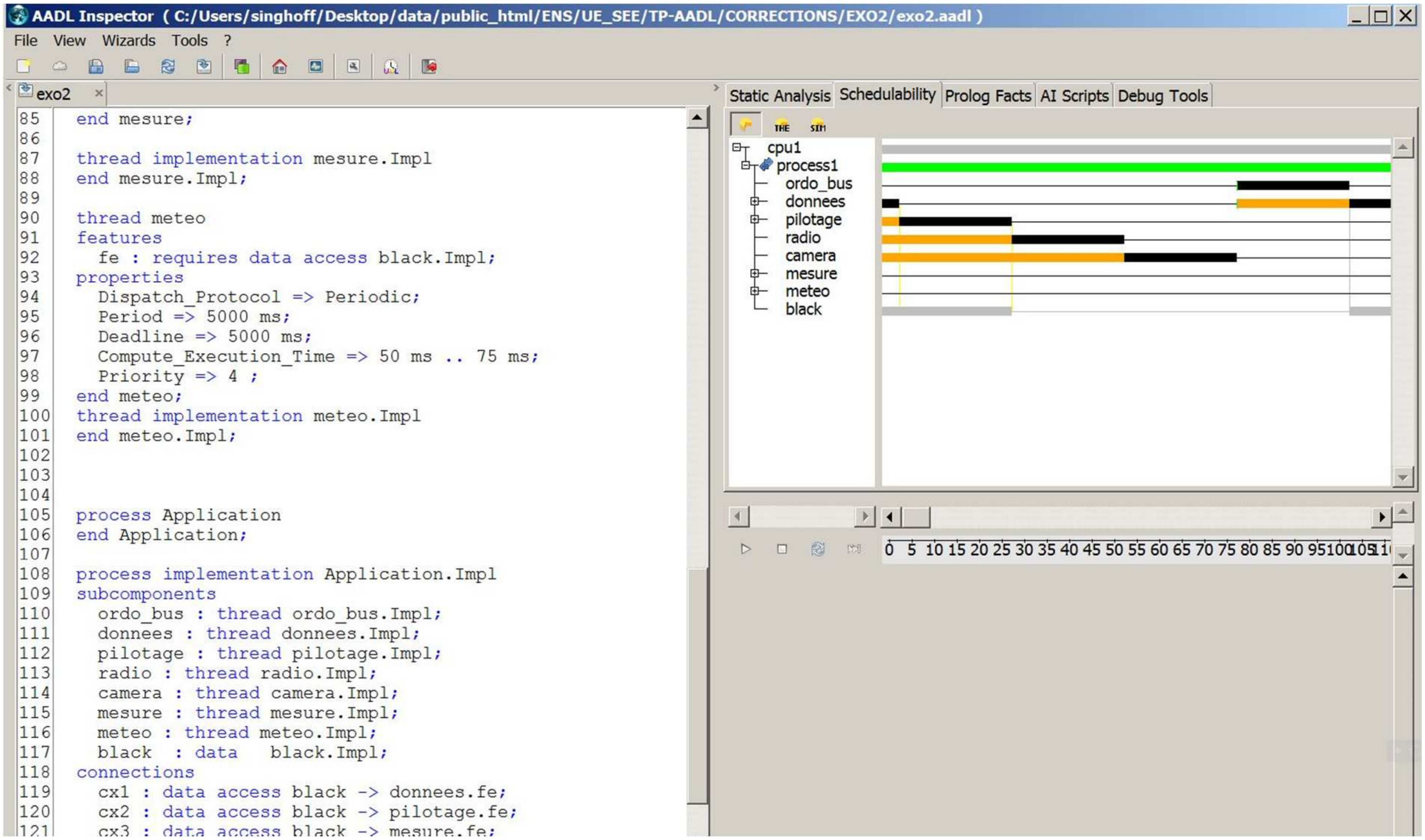

A real-time system architecture model
Evaluation result
A feasibility test applicability assumption
Platypus
Tamaris
a 'RTPatterns' mo
Simultaneous
.T.

```
DATA;
#1=PERIODIC_TASK(7, 29, 29, 0, 1, 0);
#2=PERIODIC_TASK(3, 10, 10, 0, 1, 0);
#3=PERIODIC_TASK(1, 5, 5, 0, 1, 0);
ENDSEC;
```

```
RULE Simultaneous_Release_Time FOR ( Periodic_Task );
  LOCAL
    nbpt : INTEGER := SIZEOF ( Periodic_Task );
    p1 : Periodic_Task := Periodic_Task [ 1 ];
  END_LOCAL;
WHERE
  (* All tasks share the same release time *)
  r1 : ( nbpt < 2 ) OR
   ( SIZEOF ( QUERY ( p <* Periodic_Task |
        p.Release_Time <> p1.Release_Time ) ) = 0 );
END_RULE;
```

- **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



```
  end mesure;

  thread implementation mesure.Impl
  end mesure.Impl;

  thread meteo
  features
    fe : requires data access black.Impl;
  properties
    Dispatch_Protocol => Periodic;
    Period => 5000 ms;
    Deadline => 5000 ms;
    Compute_Execution_Time => 50 ms .. 75 ms;
    Priority => 4 ;
  end meteo;
  thread implementation meteo.Impl
  end meteo.Impl;



  process Application
  end Application;

  process implementation Application.Impl
  subcomponents
    ordo_bus : thread ordo_bus.Impl;
    donnees : thread donnees.Impl;
    pilotage : thread pilotage.Impl;
    radio : thread radio.Impl;
    camera : thread camera.Impl;
    mesure : thread mesure.Impl;
    meteo : thread meteo.Impl;
    black  : data   black.Impl;
  connections
    cx1 : data access black -> donnees.fe;
    cx2 : data access black -> pilotage.fe;
    cx3 : data access black -> mesure.fe;
```

Summary

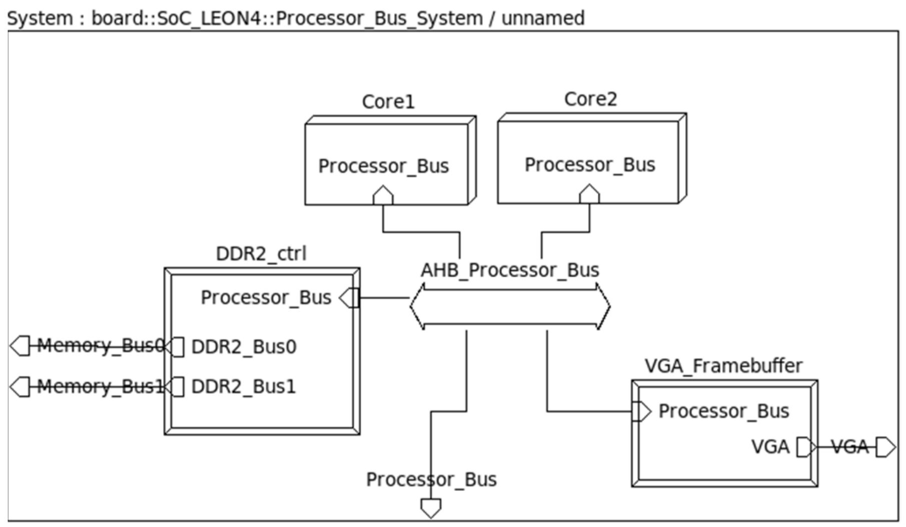
AADL Modeling of Multiprocessor Systems

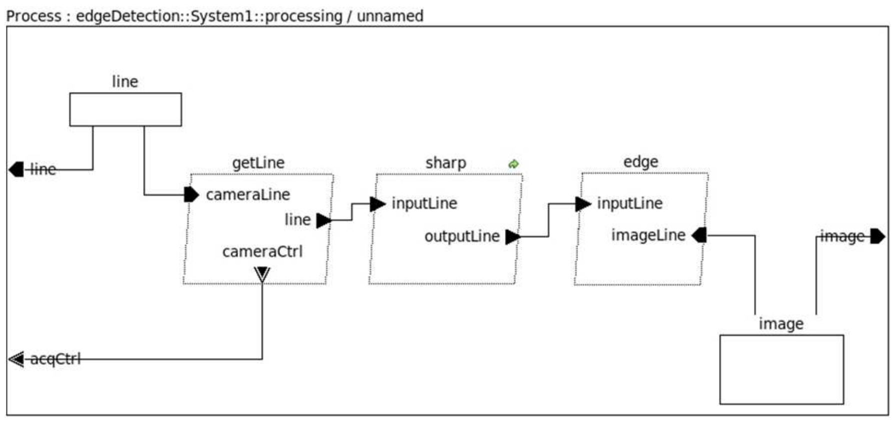
- **Problem statement:**
  1. Which design patterns for multiprocessor architectures?
  2. From scheduling analysis point of view: how to model classical multiprocessor scheduling analysis concepts:

     partitioned scheduling or global scheduling
  3. Various multiprocessor architectures?
  4. With (or without) shared resources? cache unit, NoC?
  5. Which analysis methods?
  6. Scalability, accuracy and sustainability? To enforce analysis.

- Example of a SoC with two Leon4 cores, with L1 cache and a multimedia application

Example: Design-Pattern for Partitioned Scheduling



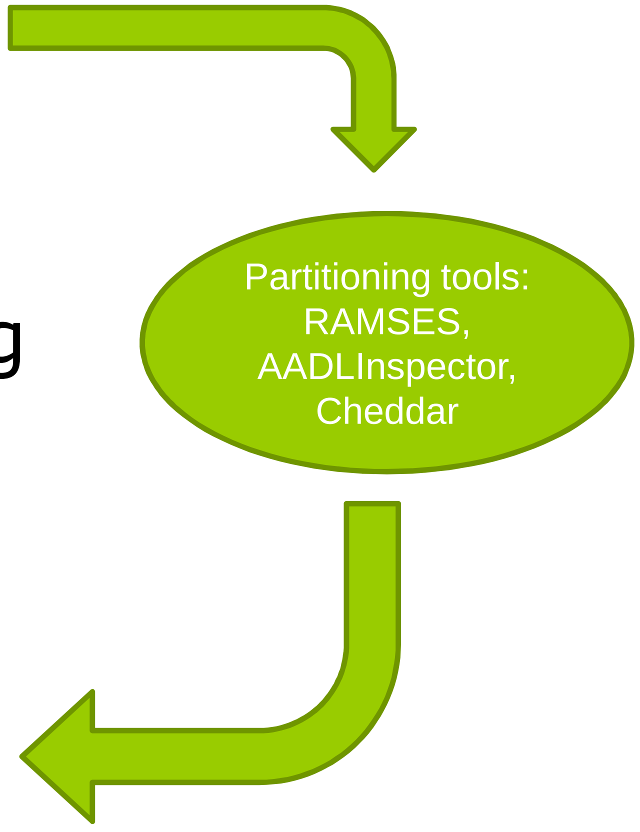


```
SYSTEM IMPLEMENTATION product.impl
SUBCOMPONENTS
  hard : SYSTEM soc_leon4::soc.asic_leon4;
  bank0 : MEMORY ram.ddr2;
  bank2 : MEMORY ram.ddr2;
  soft : PROCESS edgeDetection.impl;
PROPERTIES
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core1)) APPLIES TO soft.getLine;
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.sharp;
  actual_processor_binding => (REFERENCE(hard.Proc_System.Core2)) APPLIES TO soft.edge;
  Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core1;
  Scheduling_Protocol => (Rate_Monotonic_Protocol) applies to hard.core2;
END product.impl;
```

Design exploration, i.e. task partitioning

AADL model before partitioning

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.edge;
```

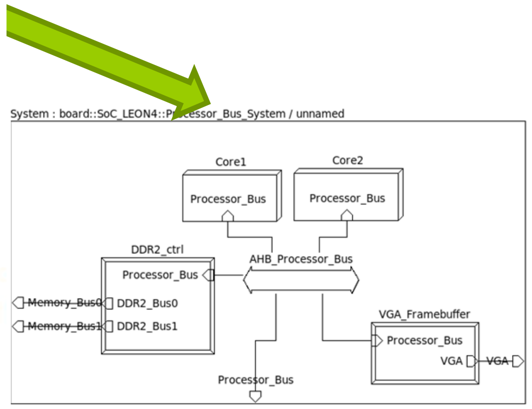


AADL model after partitioning

```
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.getLine;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.sharp;
Allowed_Processor_Binding => (REFERENCE(hard.Proc_System))
                                    APPLIES TO soft.edge;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
                                    APPLIES TO soft.getLine;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1))
                                    APPLIES TO soft.sharp;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core2))
                                    APPLIES TO soft.edge;
```

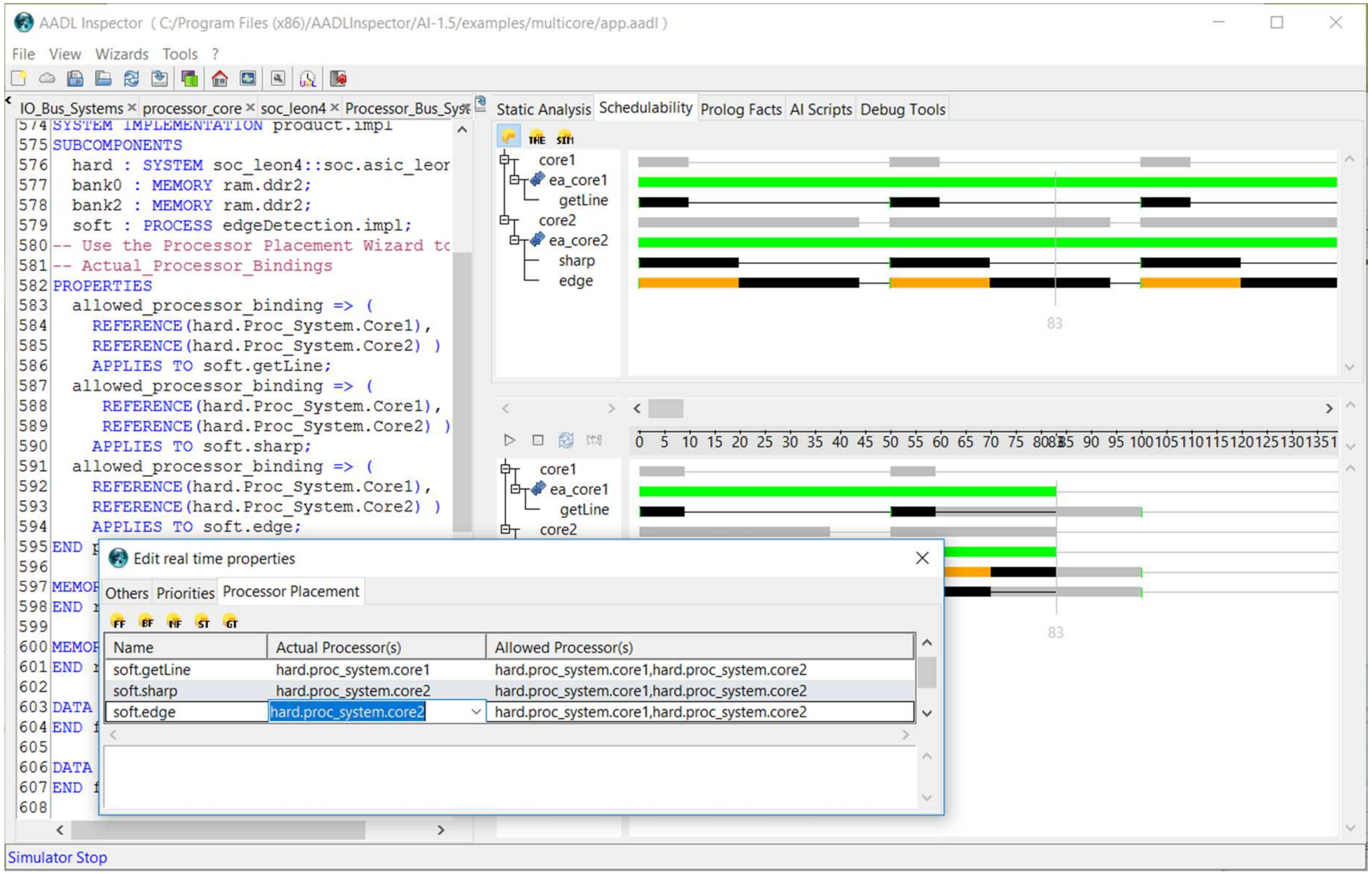
Example: Design-Pattern for Global Scheduling

```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
                             REFERENCE(hard.Proc_System.Core2))
                                  APPLIES TO soft.getLine;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
                             REFERENCE(hard.Proc_System.Core2))
                                  APPLIES TO soft.sharp;
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
                             REFERENCE(hard.Proc_System.Core2))
                                  APPLIES TO soft.edge;
Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core1;
Scheduling_Protocol => Rate_Monotonic_Protocol applies to hard.Core2;
```



```
Actual_Processor_Binding => (REFERENCE(hard.Proc_System.Core1),
                             REFERENCE(hard.Proc_System.Core2)) applies to soft;
```

Example of analysis with AADLInspector



AADL Inspector (C:/Program Files (x86)/AADLInspector/AI-1.5/examples/multicore/app.aadl)
File View Wizards Tools ?
IO_Bus_Systems processor_core soc_leon4 Processor_Bus_Syst
Static Analysis Schedulability Prolog Facts AI Scripts Debug Tools
core1
ea_core1
getLine
core2
ea_core2
sharp
edge
83
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 83 85 90 95 100 105 110 115 120 125 130 135
Edit real time properties
Others Priorities Processor Placement
FF BF NF ST GT
Simulator Stop

```
SYSTEM IMPLEMENTATION product.impl
SUBCOMPONENTS
  hard : SYSTEM soc_leon4::soc.asic_leor
  bank0 : MEMORY ram.ddr2;
  bank2 : MEMORY ram.ddr2;
  soft : PROCESS edgeDetection.impl;
-- Use the Processor Placement Wizard tc
-- Actual_Processor_Bindings
PROPERTIES
  allowed_processor_binding => (
    REFERENCE(hard.Proc_System.Core1),
    REFERENCE(hard.Proc_System.Core2) )
    APPLIES TO soft.getLine;
  allowed_processor_binding => (
     REFERENCE(hard.Proc_System.Core1),
     REFERENCE(hard.Proc_System.Core2) )
    APPLIES TO soft.sharp;
  allowed_processor_binding => (
    REFERENCE(hard.Proc_System.Core1),
    REFERENCE(hard.Proc_System.Core2) )
    APPLIES TO soft.edge;
END p

MEMOR
END r

MEMOR
END r

DATA
END f

DATA
END f
```

Name	Actual Processor(s)	Allowed Processor(s)
soft.getLine	hard.proc_system.core1	hard.proc_system.core1,hard.proc_system.core2
soft.sharp	hard.proc_system.core2	hard.proc_system.core1,hard.proc_system.core2
soft.edge	hard.proc_system.core2	hard.proc_system.core1,hard.proc_system.core2

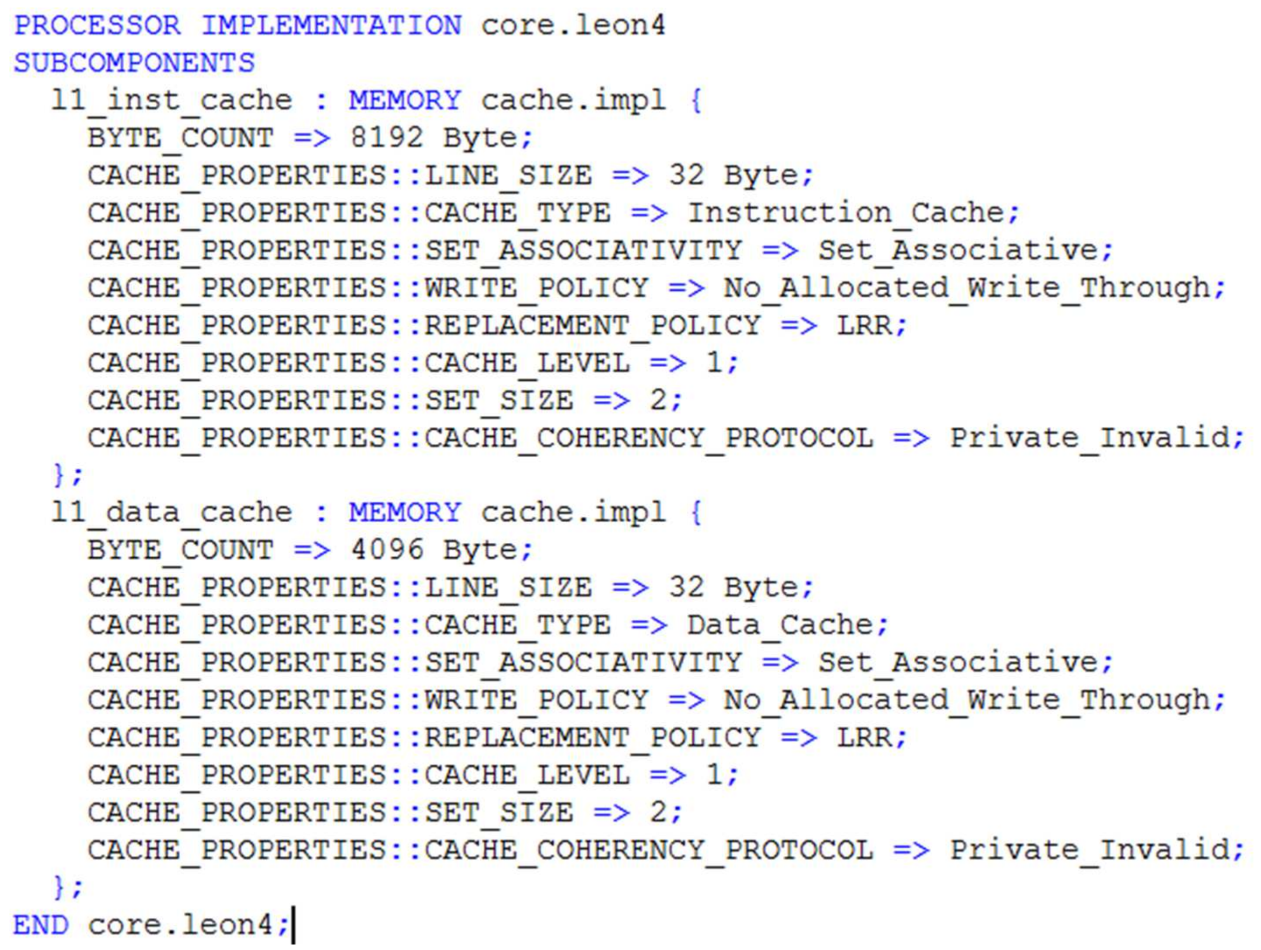
Shared resource example: cache unit

- **Problem statement:** What to model to priority assignment?
  - Thread code: to compute execution time and CFG
  - Cache and memory configuration: to compute thread cache access profile
  - Scheduling parameters

- **AADL:** is able to model thread execution time, scheduling parameters, cache and memory configuration
  - CFG? Behavior annex?
  - Cache access profile (computed)? Properties? Behavior annex?

Shared resource example: cache unit

- **Example:** an example of user-defined property sets



```
PROCESSOR IMPLEMENTATION core.leon4
SUBCOMPONENTS
  l1_inst_cache : MEMORY cache.impl {
    BYTE_COUNT => 8192 Byte;
    CACHE_PROPERTIES::LINE_SIZE => 32 Byte;
    CACHE_PROPERTIES::CACHE_TYPE => Instruction_Cache;
    CACHE_PROPERTIES::SET_ASSOCIATIVITY => Set_Associative;
    CACHE_PROPERTIES::WRITE_POLICY => No_Allocated_Write_Through;
    CACHE_PROPERTIES::REPLACEMENT_POLICY => LRR;
    CACHE_PROPERTIES::CACHE_LEVEL => 1;
    CACHE_PROPERTIES::SET_SIZE => 2;
    CACHE_PROPERTIES::CACHE_COHERENCY_PROTOCOL => Private_Invalid;
  };
  l1_data_cache : MEMORY cache.impl {
    BYTE_COUNT => 4096 Byte;
    CACHE_PROPERTIES::LINE_SIZE => 32 Byte;
    CACHE_PROPERTIES::CACHE_TYPE => Data_Cache;
    CACHE_PROPERTIES::SET_ASSOCIATIVITY => Set_Associative;
    CACHE_PROPERTIES::WRITE_POLICY => No_Allocated_Write_Through;
    CACHE_PROPERTIES::REPLACEMENT_POLICY => LRR;
    CACHE_PROPERTIES::CACHE_LEVEL => 1;
    CACHE_PROPERTIES::SET_SIZE => 2;
    CACHE_PROPERTIES::CACHE_COHERENCY_PROTOCOL => Private_Invalid;
  };
END core.leon4;
```

Cache/CRPD-Aware Priority Assignment Algorithm

- ❑ In fixed priority preemptive scheduling context, tasks can preempt and evict data of other tasks in the cache.
- ❑ Cache related preemption delay **(CRPD)**: additional time to refill the cache with the cache blocks evicted by the preemption.

- ❑ **Problem statement:**
  - ❑ CRPD may be high, non-negligible preemption cost (Pellizzoni et al., 2007).
  - ❑ No fixed priority assignment algorithm takes CRPD into account.

Cache/CRPD-Aware Priority Assignment Algorithm

- **Approach:**
  - Extend Audsley's priority assignment algorithm (Audsley, 1995) to take into account CRPD.
  - CRPD-aware priority assignment algorithms (**CPA**) that assign priority to tasks and verify theirs schedulability.
  - 5 algorithms with different levels of schedulability efficiency (1) and complexity (2), scalability (3).
  - Implemented into Cheddar, Not available in AADLInspector today

	CPA-PT-Simplified	CPA-PT	CPA-Tree	Exhaustive Search
(1)	0.65	0.72	0.80	0.87
(2)	Low	Medium	High	
(3)	100 tasks	30 tasks	10 tasks	

Scheduling Simulator with execution platform shared resources

- **Problem statement:**
  - How to model shared resources? Abstraction level?
  - Tick accurate versus cycle accurate
  - Take into account various parameters from different sources (tools)
  - Theoretical issues (feasibility interval, sustainability, accuracy)

- Focus on instruction cache, NoC
- Not available in AADLInspector today

- **Example with NoC:**
  - Dual Task and Flow Model (DTFM) : computes the flow model from a task model, task mapping, precedence constraints on a NoC
  - Identify/compute delays induced by Wormhole XY NoC and perform scheduling analysis for AADL data port software design
  - Evaluation with tick + cycle accurate simulator (SHOC SystemC NoC simulator + Cheddar)

Summary

Conclusion

- **AADL & early scheduling analysis:**
  - Design-pattern approach
  - Constraints on model entities (e.g. component categories, feature categories, properties, annexes) to enforce analysis
  - One model for a given intend
  - Examples of both software and execution platform modeling/analysis
  - Current work: multiprocessor architectures
  - Evaluation, details => see publications

- **Lessons learnt:**
  - AADL v2 is enough to model (most of) the software/execution platforms we investigated
  - Tools can interact: STOOD, RAMSES, AADLInspector, Ocarina, OSATE, WCET tools, System-C simulators, Cheddar, …
  - Analysis features developed by the Lab-STICC: prototyped with Cheddar, available for AADL with AADLInspector