NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

TELECOM ParisTech
Une école de l'IMT

LTCI
Laboratoire Traitement et Communication de l'Information

Architecture, illustrated with AADL
Architecture Analysis & Design Language

MPM4CPS training school

Etienne Borde
etienne.borde@telecom-paristech.fr

General context of CPS

- CPS : the cyber part is becoming more and more complex because of
  - Hardware capabilities and, as a consequence, software complexity
  - Size of the design space
  - Security and safety requirements
  - Lack of well accepted modeling techniques for CPS
- Shorten time-to-market
  - Requires to automate parts of the design process (e.g. analysis, documentation, code generation…)
  - Needs for reuse of existing functions (legacy software/hardware/network)
- Focus of this presentation: AADL, a suitable architecture description language for the cyber part of CPS:
  - Ease documentation of the design
  - Improve designers team communication
  - Analyze properties and verify requirements (functional or non-functional)
  - Automate model transformations and source code generation

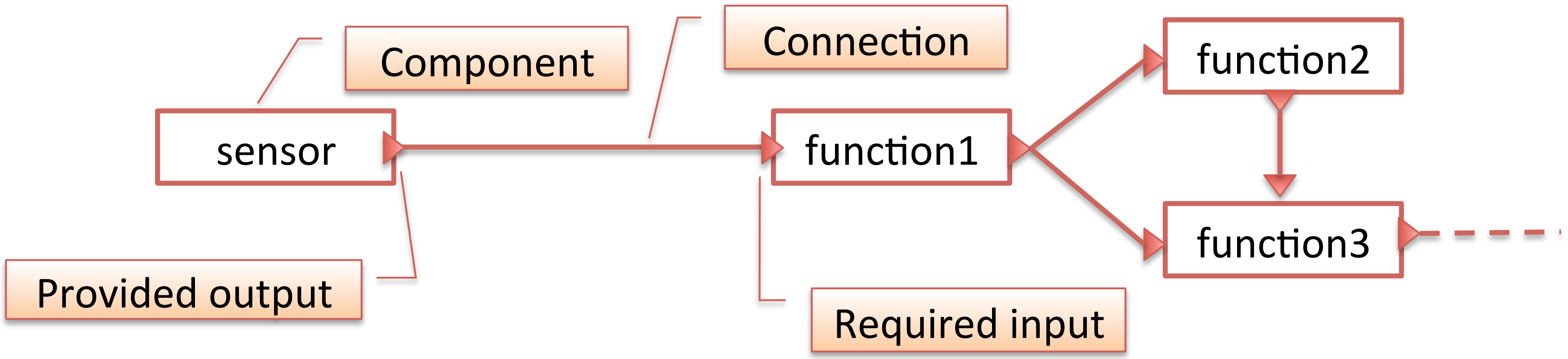
Models definition

Modeling, in the broadest sense, is the ***cost-effective*** *use of something in place of something else for some* ***cognitive purpose****. It allows us to* ***use something that is simpler, safer or cheaper than reality instead of reality for some purpose****. A model represents reality for* ***the given purpose****; the model is an* ***abstraction of reality in the sense that it cannot represent all aspects of reality****. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.*

"The Nature of Modeling" Jeff Rothenberg in Artificial Intelligence, Simulation, and Modeling, L.E. William, K.A. Loparo, N.R. Nelson, eds. New York, John Wiley and Sons, Inc., ***1989****, pp. 75-92*

Models in computer science

- In computer science, architecture models are meant to represent the **organisation of a computer system** as a **set of components** and their **interactions**
- Main artefacts: boxes and arrows (Woaouh!...)
  - ⌘ Components : main elements of the design
  - ⌘ Interfaces : what components offer and what they need
  - ⌘ Connections : resolve components needs



- Then drawing becomes programming… or at least designing… Nothing new conceptually.
- What about the semantic?

Architecture Description Languages (ADL) - categories

- Formal ADLs, i.e. base on a mathematically sound definition of their semantic
  - Meant to formally verify/prove expected properties of a computer system
  - Wright, Data flow graphs, state machines, …
- Domain specific ADLs
  - Meant to describes the design and implementation of computer systems constituents
  - UML 2, AUTOSAR, ***AADL***
- Abstract ADLs
  - Meant to describe the organization of a computer system without providing a precise semantics
  - ArchJava, Fractal

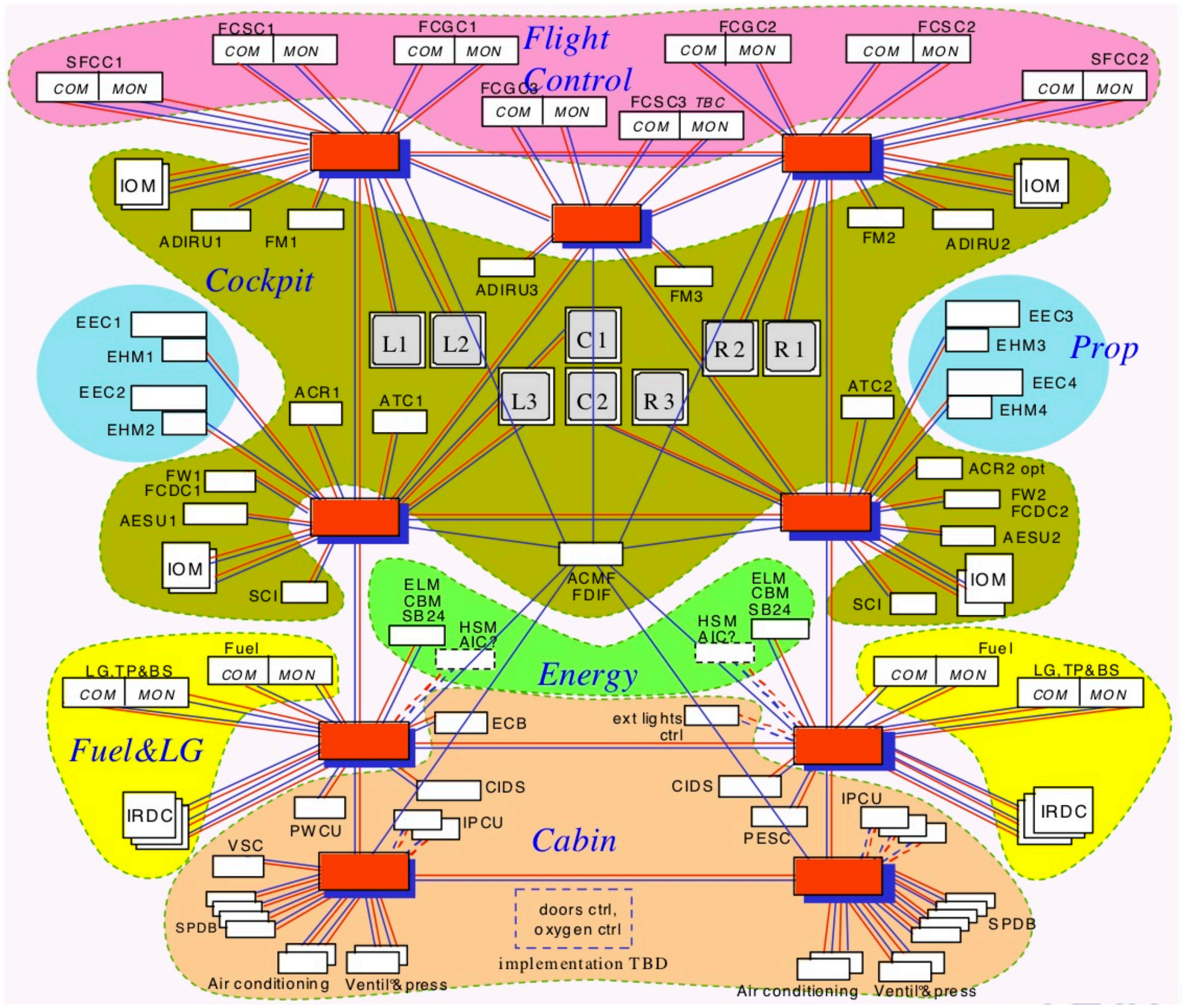
Note: some ADLs are standardised (e.g. UML, AADL), which provides a common understanding of the notation to the cost of slow evolutions through a committee.

Software modeling languages in industry

- UML is used in software engineering in general, but not necessarily for real-time embedded systems (e.g. Simulink, SCADE, AUTOSAR, AADL…)

- Main industrial use, in order of use:
  - ⌘ Specification,
  - ⌘ Documentation,
  - ⌘ Design,
  - ⌘ Analysis/verification,
  - ⌘ Code generation

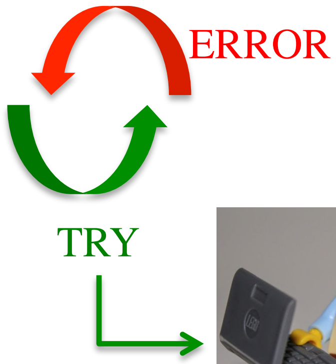
Exemple of a big real-time system architecture

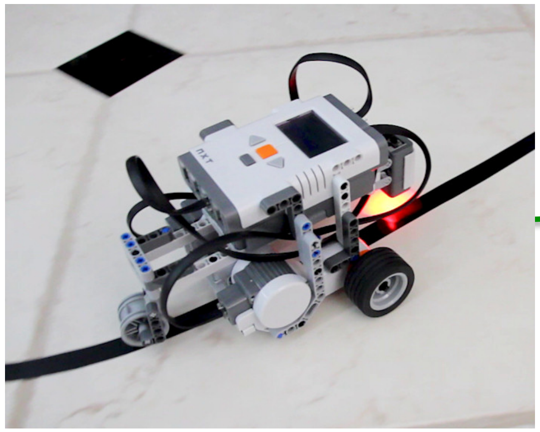


- Explain me the paradigm…

Running example

- Initial plan: a really complex system





→ It works!

- Actual case-study :
  a bit smaller, but still challenging and representative

What is AADL (Architecture Analysis and Design Language)?

- An ADL as well as a DSL for real-time embedded systems:
  - ⌘ Uses the principles of a concrete DSL (components, interfaces and connections)
  - ⌘ Define properties for real-time and embedded systems analysis
    - o Scheduling policy, compute execution time, latency…
    - o Software components to hardware components allocation

- Objective: assist the design of such systems
  - ⌘ Standardized semantics (formulated with natural language)
  - ⌘ Textual and graphical syntax
  - ⌘ Strongly typed (components category, composition rules, …)
  - ⌘ Extendable (property definition language and annexes)

AADL overview

General characteristics

- Components are the main modeling entities
  - ⌘ Belong to a set of predefined categories
- The standard defines categories of components (keywords of the language); examples of categories a component can belong to:
  - ⌘ System
  - ⌘ Process, thread, data…
  - ⌘ Processor, memory, bus…
- Components definition is divided into types, implementations, and subcomponents
  - ⌘ Type: how the component is viewed from outside (e.g. interaction interfaces)
  - ⌘ Implementation: internal structure of the component (e.g. subcomponents)
  - ⌘ Subcomponents: instances of components, starting from a root system implementation
- Characteristics of components are structured into sections (e.g. **features, properties, subcomponents**) identified by keywords of the language

- Details:
  - ⌘ Components can be declared in any order
  - ⌘ The language is case insensitive

System level viewpoint: categories

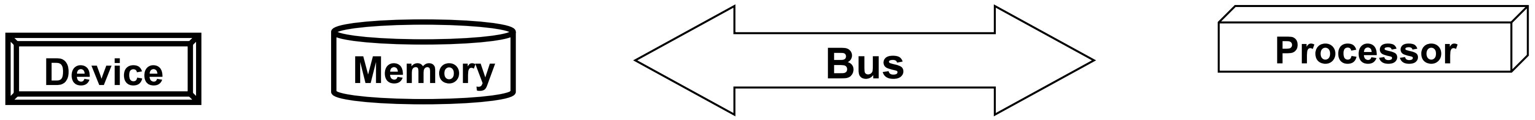
- Two categories: system and abstract

System
Abstract

- Different possible objectives
  - ⌘ Represent, from a very abstract view point, the main constituent of the system, their interfaces and connections.
  - ⌘ System:
    - ○ aggregate, by composition, subcomponents describing the execution platform and subcomponents describing the software architecture.
    - ○ Define the main operational modes of the system
  - ⌘ Abstract:
    - ○ Define structure and interaction without knowing yet the nature of the component

Execution platform viewpoint: categories

- Instances of components categories dedicated to the execution platform description
  - processor : hardware computation unit + tasks scheduling capabilities
  - memory : storage component (may be RAM, hard drive disk, cache, etc.)
  - bus : physical communication link (network cable, etc.)
  - device : interface with the physical environment of the system (sensors/ actuators)

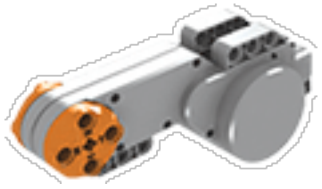


Illustrative example of execution platform components categories

Color Sensor

NXT_CPU

Motor

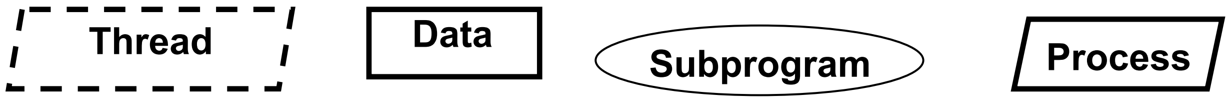


Bus

Sonar

Software architecture viewpoint: categories

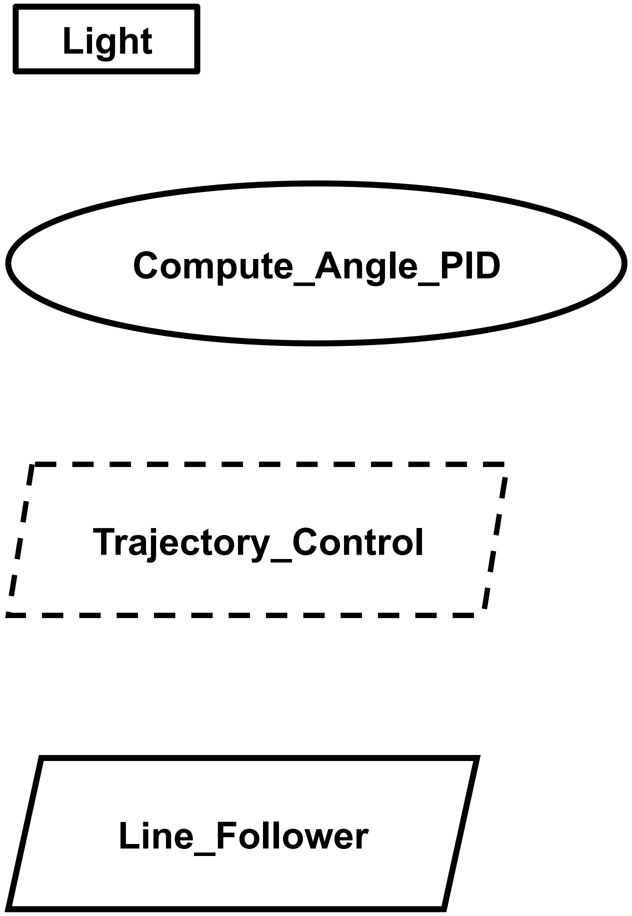
- AADL component categories for software
  - ⌘ Data : information that can be exchanged among software components (not the same focus as data types in programming languages)
  - ⌘ Subprogram: sequentially executable software, like functions in C programming language
  - ⌘ thread : task executing a sequence of functions
  - ⌘ process : memory address space allocated for the execution of its thread subcomponents



- These categories focus on operating system and programming elements

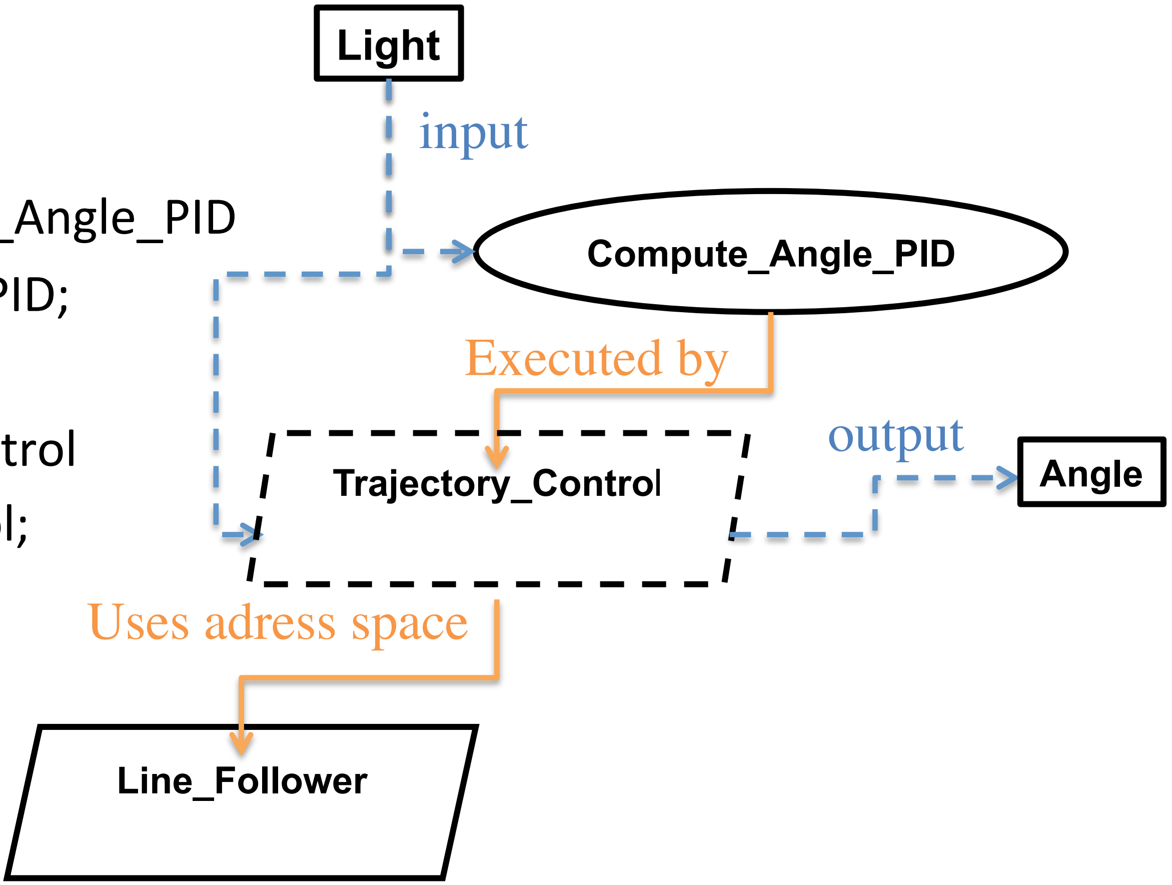
Example of software components

```
data Light
end Light;

subprogram Compute_Angle_PID
end Compute_Angle_PID;

thread Trajectory_Control
end Trajectory_Control;

process Line_Follower
end Line_Follower
```



Example of software components

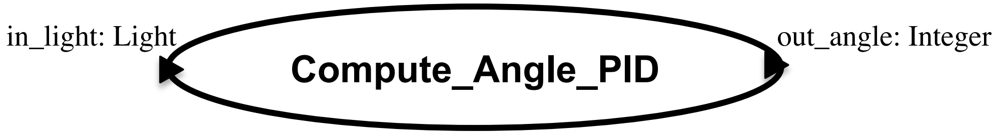
```
data Light
end Light;

subprogram Compute_Angle_PID
end Compute_Angle_PID;

thread Trajectory_Control
end Trajectory_Control;

process Line_Follower
end Line_Follower
```



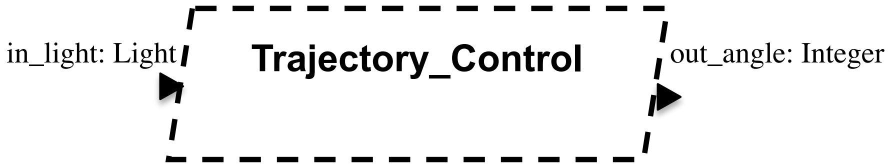
How to represent these interactions and allocations in AADL?

First, we need to define component interfaces (called features in AADL)

- Parameters
  - ⌘ **in**, **out**, or **inout**
  - ⌘ Usable for subprograms



- Requires or provides data access
  - ⌘ Usable for subprograms and threads
- Ports
  - ⌘ **in**, **out**, or **in out**
  - ⌘ **Data**, **event** or **event data**
  - ⌘ Usable for threads and processes

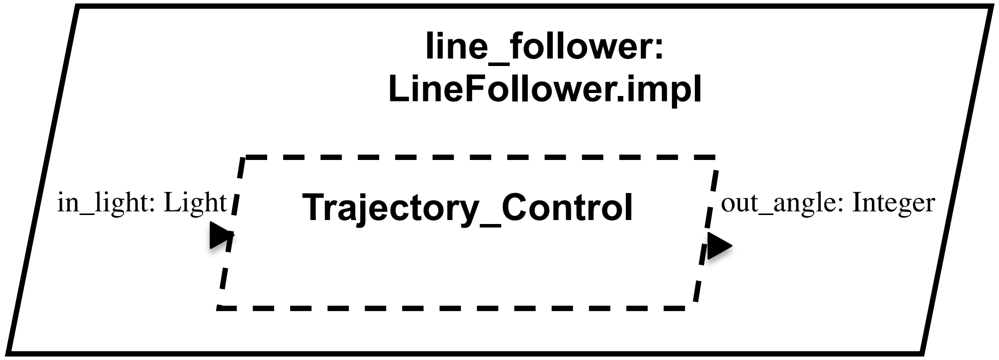


Semantical differences among AADL features of software components

- Data port versus event data port
  - ⌘ **data port :** single value shared among components (no queueing).
  - ⌘ **event** or **event data port** : multiple values queued.
- Data port versus Data access
  - ⌘ Data access allows access to the data at anytime during the execution of a task/ subprogram
  - ⌘ Data port defines the following semantics:
    - ○ Data becomes available on an input port when the thread starts its execution. Data not used in the previous execution of the thread is lost. Data is not updated during the execution of the task.
    - ○ Data produced on an output port are sent to the recipient port at the end of the producer task.

Second, we need to compose

- Thread subcomponents in process implementations:



```
process implementation Proc.simple
   subcomponents
      C_Th   : thread ContrThread.Impl;
end Proc.simple;
```

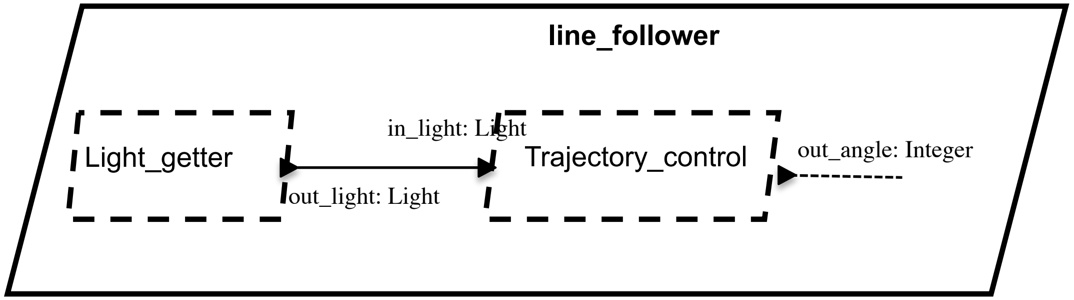
- Subprogram calls in call sequences

```
thread implementation ContrThread.Impl
calls
   call1:
   {
     cp   : subprogram computePID;
     cs   : subprogram computeSpeed;
   };
end ContrThread.Impl;
```

Sequence of subprogram calls

Third, we need to connect

- Components features are connected hierarchically
  - Thread subcomponents in the process



```
process implementation Proc.simple
   subcomponents
      C_Th   : thread ContrThread.Impl;
   connections
      c2: port Bg_Th.out_light -> C_Th.in_light;
end Proc.simple;
```

  - Subprogram calls in the thread but also the thread features with the subprogram calls

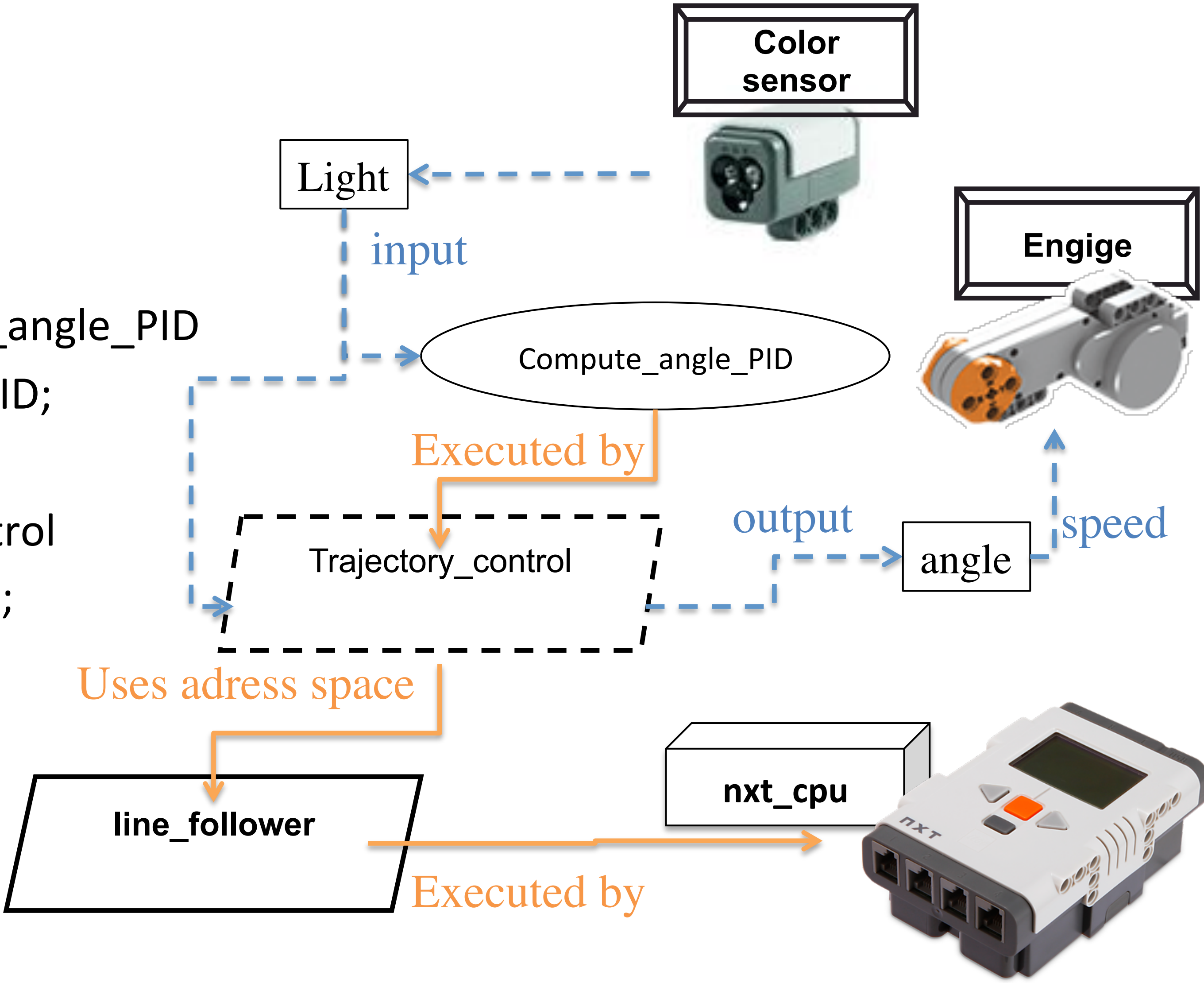
```
thread implementation ContrThread.Impl
calls
   call1:
   {
     cp   : subprogram computePID;
     cs   : subprogram computeSpeed;
   };
connections
   cc0 : parameter cp.currentLight -> in_light;
   cc1 : parameter cp.angle -> cs.angle;
end ContrThread.Impl;
```

Value received on the input port is passed as a parameter to the subprogram

Value produced by computePID is passed to compute Speed

How to combine execution platform and software architecture viewpoints in AADL

```
data Light
end Light;

subprogram compute_angle_PID
end compute_angle_PID;

thread trajectory_control
end trajectory_control;

process line_follower
end line_follower
```



A few words about properties

- Properties in AADL is a way to decorate your model
  - ⌘ Properties can be associated to almost any element of your model.
  - ⌘ The standard defines a property language: AADL users can define their own **property sets** (means to extend the possibilities of the language).
  - ⌘ The AADL standard predefines a set of properties for most common used properties in ADLs and real-time embedded systems.
- Example of property definition

```
Actual_Processor_Binding: inherit list of reference (processor,
                                                     virtual processor,
                                                     system, device)
                 applies to (thread, thread group, process,
                            system, virtual processor, device);
```

- Example of property association

```
SYSTEM IMPLEMENTATION synchronous.others
  SUBCOMPONENTS
    my_platform : PROCESSOR CPU;
    my_process  : PROCESS my_process.impl;
  PROPERTIES
    Actual_Processor_Binding => ( reference(my_platform) ) applies to my_process;
END synchronous.others;
```

Things I could not present

- AADL has much more than this
  - ⌘ Lots of standardized properties
  - ⌘ Modes, with mode-specific architectures (reconfiguration)
  - ⌘ Flows, to analyse the worst-case latency and jitter of data
  - ⌘ System/Hardware/Network configuration representation
  - ⌘ Behavior as state machines
  - ⌘ Errors, faults, propagation

NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

TELECOM ParisTech
Une école de l'IMT

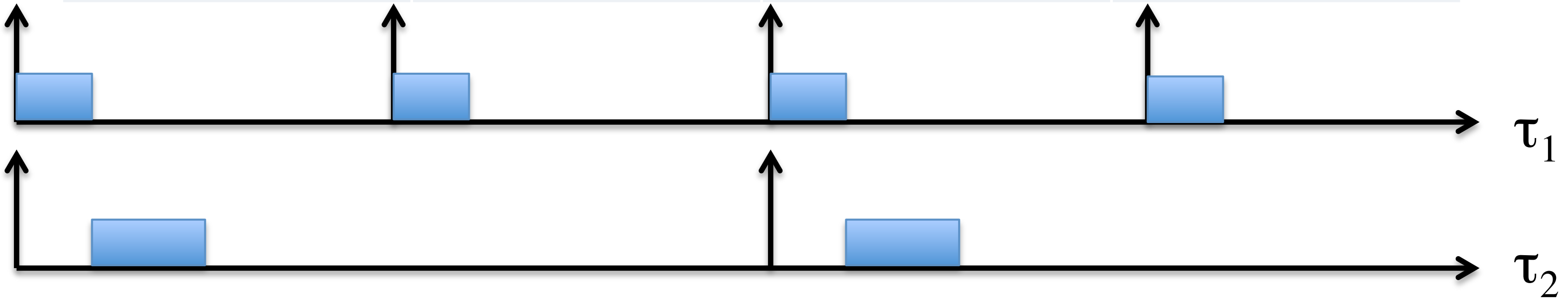
LTCI
Laboratoire Traitement et Communication de l'Information

Using AADL for Response Time Analysis

Principles of Response Time Analysis (RTA)

- Problem we are trying to anser: the computation unit is shared among different tasks… Is that safe from a timing performance viewpoint?
  - Expressed as: tasks never miss their deadlines
- To simplify, we assume a set of periodic tasks with a fixed priority scheduling configured using the Rate Monotonic Scheduling principles
  - RMS: the higher the frequency, the higher the priority
- RTA is meant to compute the worst-case response time of a task, based on:
  - The tasks period
  - The tasks worst case execution time (WCET) for the code of the task
  - The tasks deadline

Task	Period	WCET	Deadline
τ_1	10 ms	2 ms	6 ms
τ_2	20 ms	3 ms	10 ms



AADL Properties for response time analysis

- In terms of structure, we need
  - Threads, representing tassk
  - The processor on which threads are deployed
  - The property value assigning threads to a processor
- Scheduling properties are
  - *Dispatch_protocol*, usually set with value ***periodic***
  - *Compute_execution_time* represents the execution time interval, where the upper bound is the WCET
  - The *Priority*, meaning is obvious.
  - *Deadline*, meaning is obvious.
  - *Scheduling_Protocol*, associated to the processor, defines the scheduling policy applied by the operating system running on the processor

Exemple (1/2)

```
PACKAGE synchronous_Pkg
PUBLIC
WITH Base_Types;

SYSTEM synchronous
END synchronous;

SYSTEM IMPLEMENTATION synchronous.others
  SUBCOMPONENTS
    my_platform : PROCESSOR CPU;
    my_process : PROCESS my_process.impl;
PROPERTIES
Actual_Processor_Binding =>
         ( reference(my_platform) )
           applies to my_process;
END synchronous.others;
```

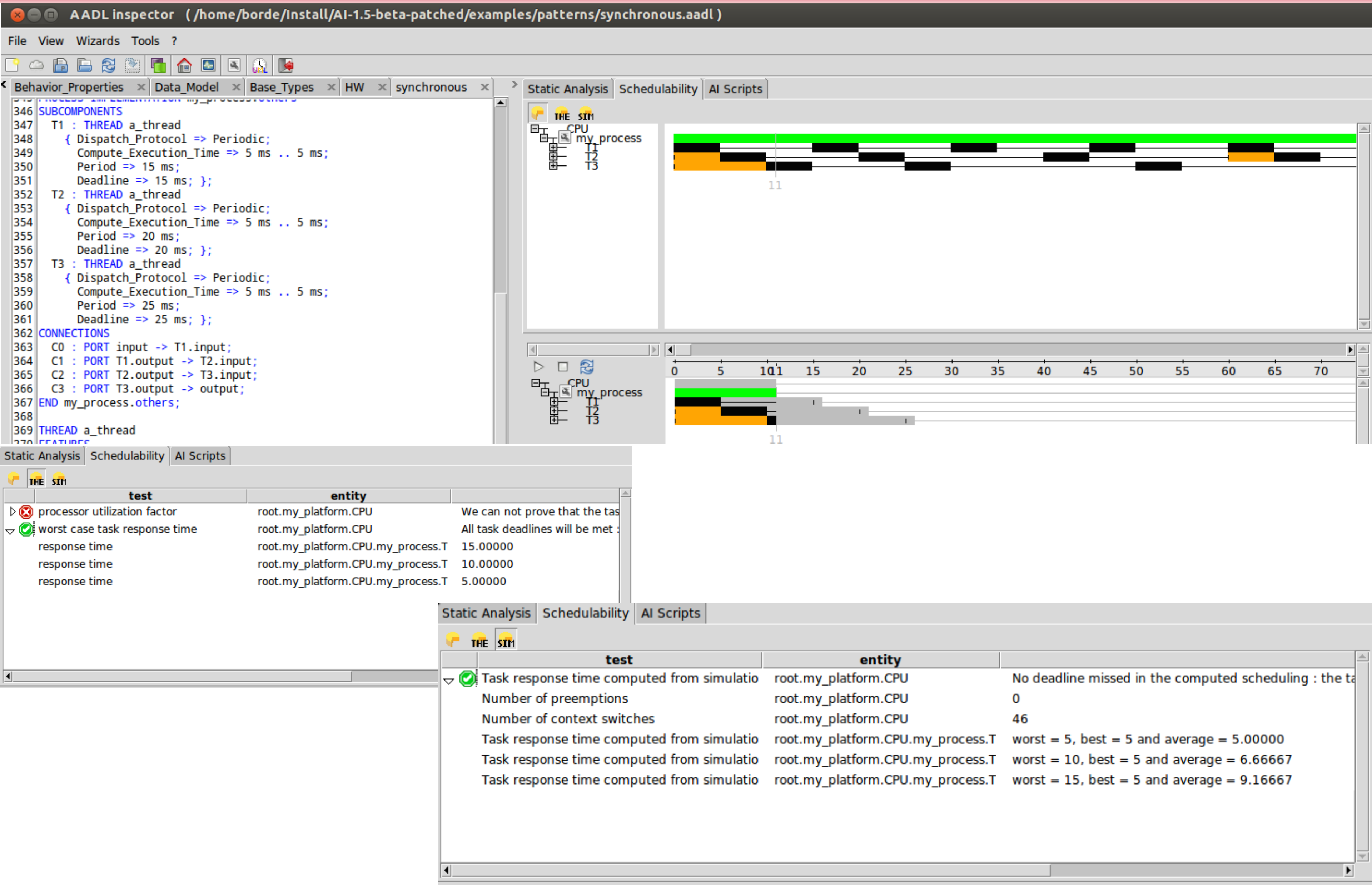
```
PROCESSOR CPU
PROPERTIES
    Scheduling_Protocol => (RMS);
END CPU;


PROCESS my_process
END my_process;
```

Exemple (2/2)

```
PROCESS IMPLEMENTATION my_process.impl
SUBCOMPONENTS
T1 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 15 ms;
     Deadline => 15 ms; };
T2 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 20 ms;
     Deadline => 20 ms; };
T3 : THREAD a_thread
   { Dispatch_Protocol => Periodic;
     Compute_Execution_Time=>5 ms..5 ms;
     Period => 25 ms;
     Deadline => 25 ms; };
END my_process.impl;
```

```
THREAD a_thread
END a_thread;


END synchronous_Pkg;
```

Exploitation avec AADL Inspector



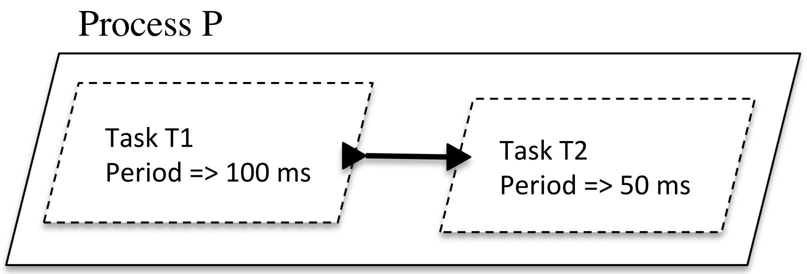
```
SUBCOMPONENTS
  T1 : THREAD a_thread
    { Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 5 ms .. 5 ms;
      Period => 15 ms;
      Deadline => 15 ms; };
  T2 : THREAD a_thread
    { Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 5 ms .. 5 ms;
      Period => 20 ms;
      Deadline => 20 ms; };
  T3 : THREAD a_thread
    { Dispatch_Protocol => Periodic;
      Compute_Execution_Time => 5 ms .. 5 ms;
      Period => 25 ms;
      Deadline => 25 ms; };
CONNECTIONS
  C0 : PORT input -> T1.input;
  C1 : PORT T1.output -> T2.input;
  C2 : PORT T2.output -> T3.input;
  C3 : PORT T3.output -> output;
END my_process.others;

THREAD a_thread
```

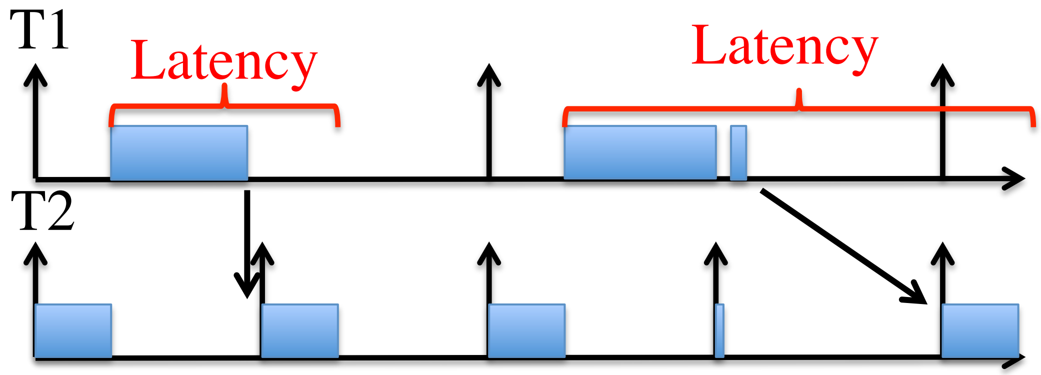
test	entity	
processor utilization factor	root.my_platform.CPU	We can not prove that the tas
worst case task response time	root.my_platform.CPU	All task deadlines will be met :
response time	root.my_platform.CPU.my_process.T	15.00000
response time	root.my_platform.CPU.my_process.T	10.00000
response time	root.my_platform.CPU.my_process.T	5.00000

test	entity	
Task response time computed from simulatio	root.my_platform.CPU	No deadline missed in the computed scheduling : the ta
Number of preemptions	root.my_platform.CPU	0
Number of context switches	root.my_platform.CPU	46
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 5, best = 5 and average = 5.00000
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 10, best = 5 and average = 6.66667
Task response time computed from simulatio	root.my_platform.CPU.my_process.T	worst = 15, best = 5 and average = 9.16667

Communications through data ports

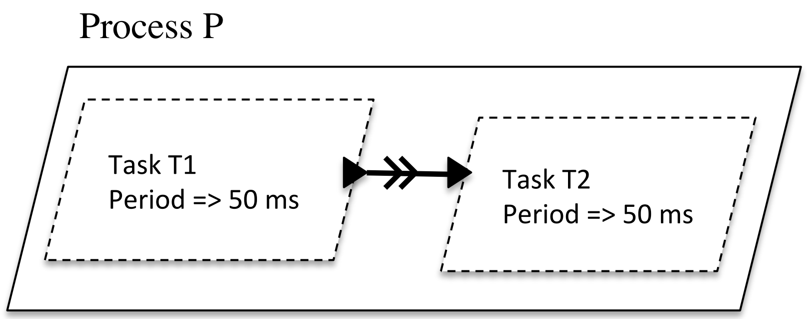
- Configured with the Timing property; possible values:
  - Sampled (default): similar to a shared variable, except for the read/execute/write semantics
    - Advantage: simplicity
    - Disadvantage: undeterministic

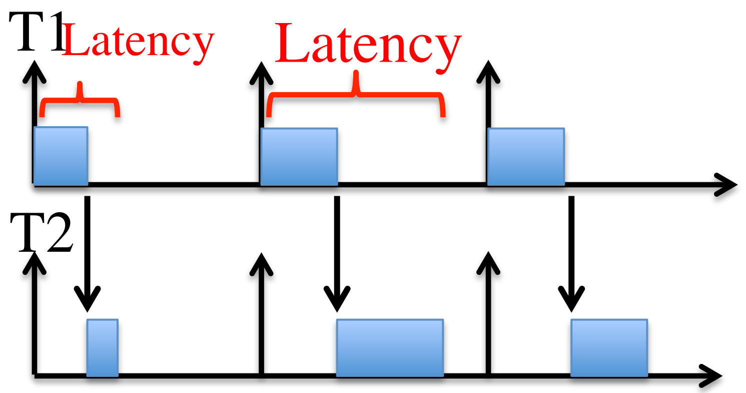




Communications through data ports

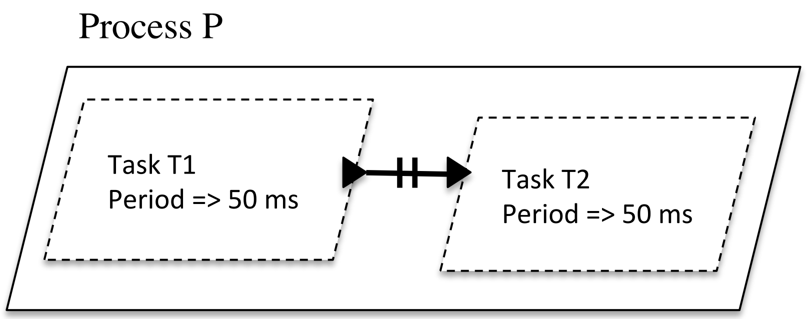
- Configured with the Timing property; possible values:
  - ⌘ Immediate: the recipient is not supposed to start until the output port of the connected thread has been updated
    - ○ Advantages: deterministic, reduces latency
    - ○ Disadvantages: put constraints on the scheduler and the model (no cycle, consistent periods)

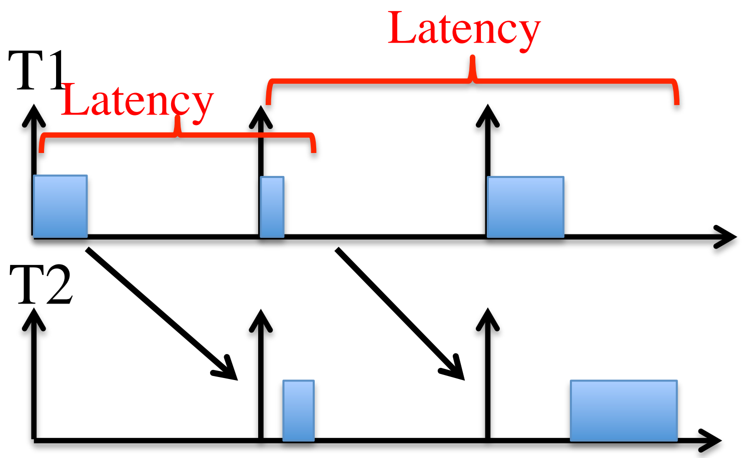




Communications through data ports

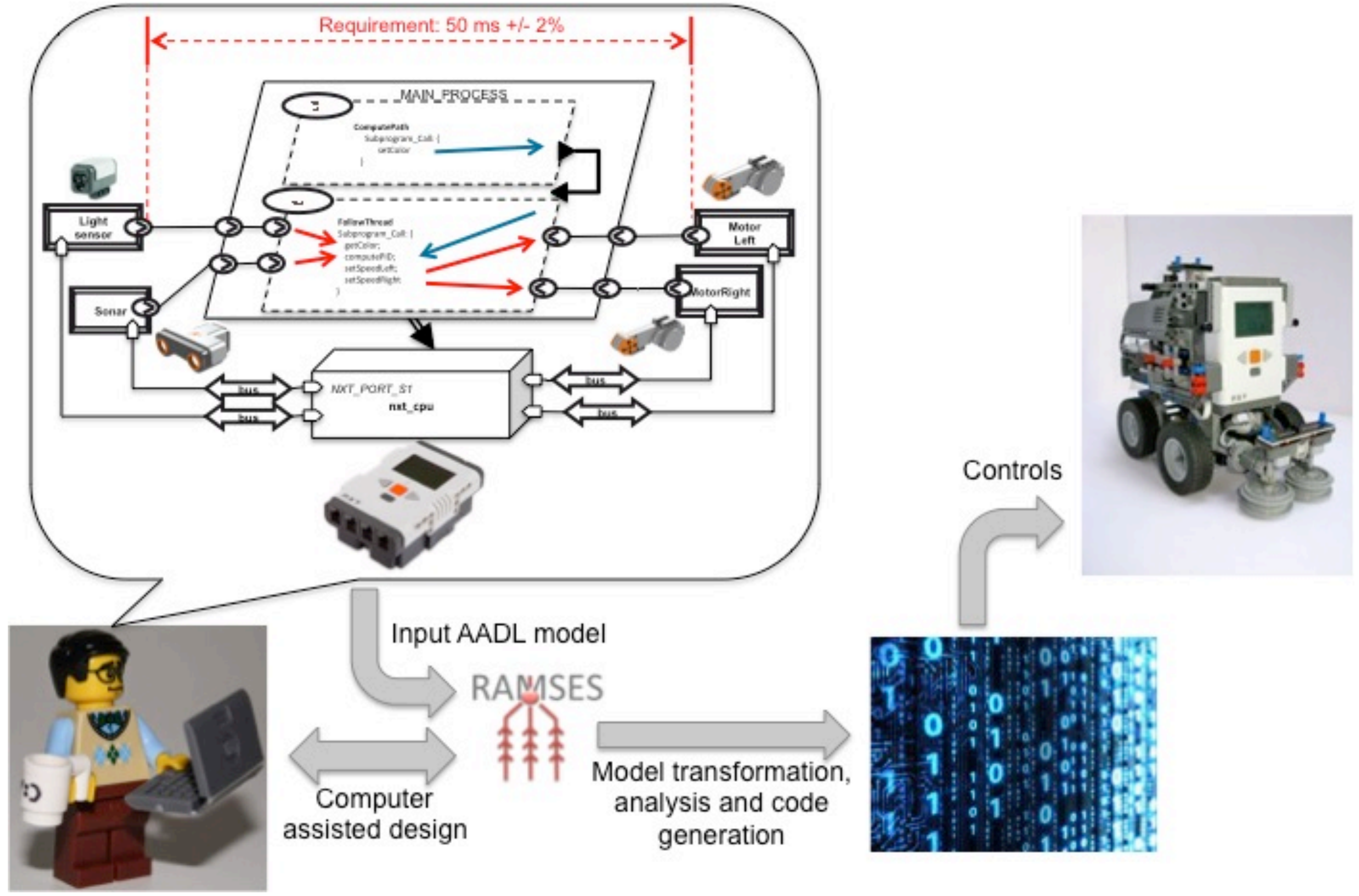
- Configured with the Timing property; possible values:
  - ⌘ Delayed: the output port is updated at the deadline of its thread
    - ○ Advantages: deterministic, reduces jitter
    - ○ Disadvantages: increases latency





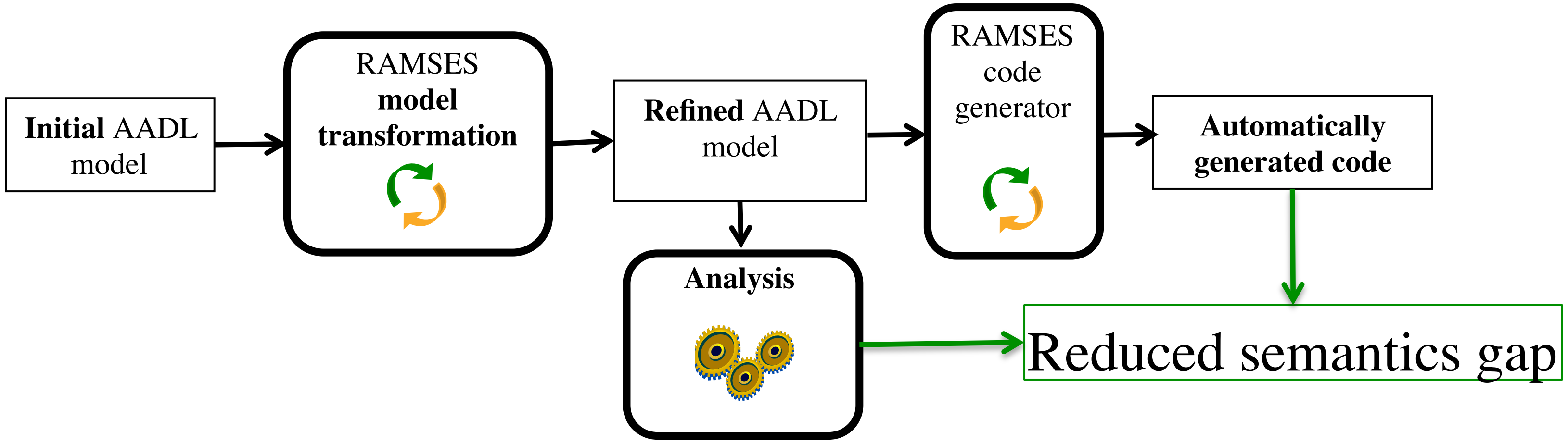
Objective of the assignment

Understand how to model resource sharing to
1 – make sure it does not jeopardize the program execution because of performance issues
2 – define queue size of communicating tasks
3 – integrate new functions such as an obstacle detection task



A word about RAMSES

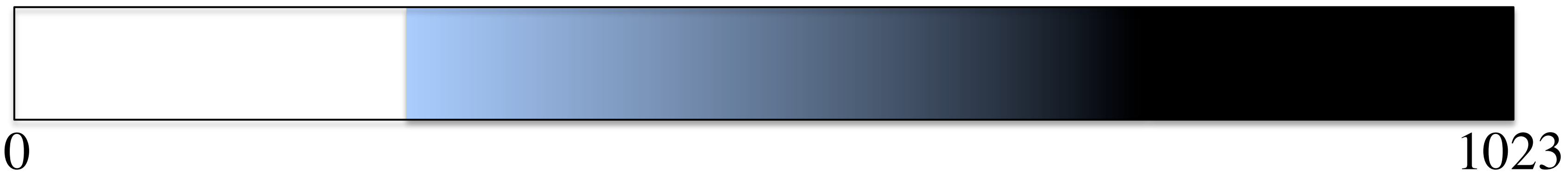
- RAMSES: Refinement of AADL Models for Synthesis of Embedded Systems

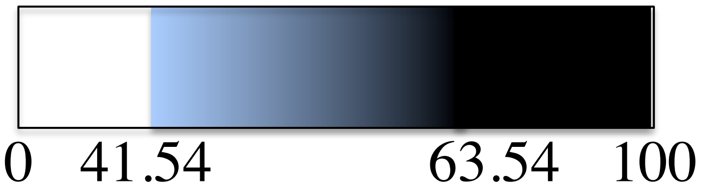


- PAPER(S) on model refinements for code generation
  - ⌘ Scheduling Multi-Periodic Mixed Criticality DAGs on Multi-Core Architectures. In 39th IEEE Real-Time Systems Symposium (RTSS2018). To be published, accepted in August 2018.
  - ⌘ Availability enhancement and analysis for mixed-criticality systems on multi-core. Design Automation and Test in Europe (DATE 2018)
  - ⌘ Multi-objectives Refinement of AADL Models for the Synthesis Embedded Systems (mu-RAMSES). ICECCS 2015
  - ⌘ Architecture Models Refinement for Fine Grain Timing Analysis of Embedded Systems. IEEE International Symposium on Rapid System Prototyping (RSP'14)
  - ⌘ PDP 4PS : Periodic-Delayed Protocol for Partitioned Systems. Ada-Europe 2014
  - ⌘ Deterministic implementation of periodic-delayed communications and experimentation in AADL. ISORC 2013
  - ⌘ Design Patterns for Rule-based Refinement of Safety Critical Embedded Systems Models. International Conference on Engineering of Complex Computer Systems (ICECCS'12)

Links with previous execrices

- The work focuses on the deployment problem for CPS, looking at the impact of resources sharing
- We used the methods introduced in the control session, more particularly the ziegler-nichols method to define the parameters of the PID
  - ⌘ it requires lots of testing → we do not ask you to reproduce this
  - ⌘ set value (we want to obtain) is set to 56 after rescaling possible light values range is [0; 100]





$(63.54 + 41.54)/2 = 53$

Trainers team (some years ago)

João Cambeiro (FCT)

Control and network engineer

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Etienne Borde, (TPT)

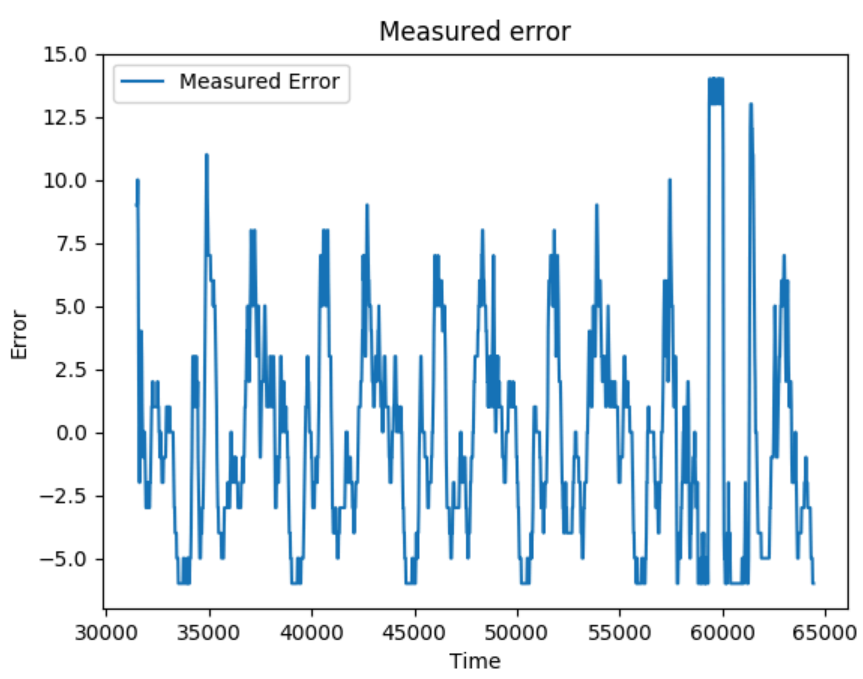
Real-time systems engineer

TELECOM ParisTech

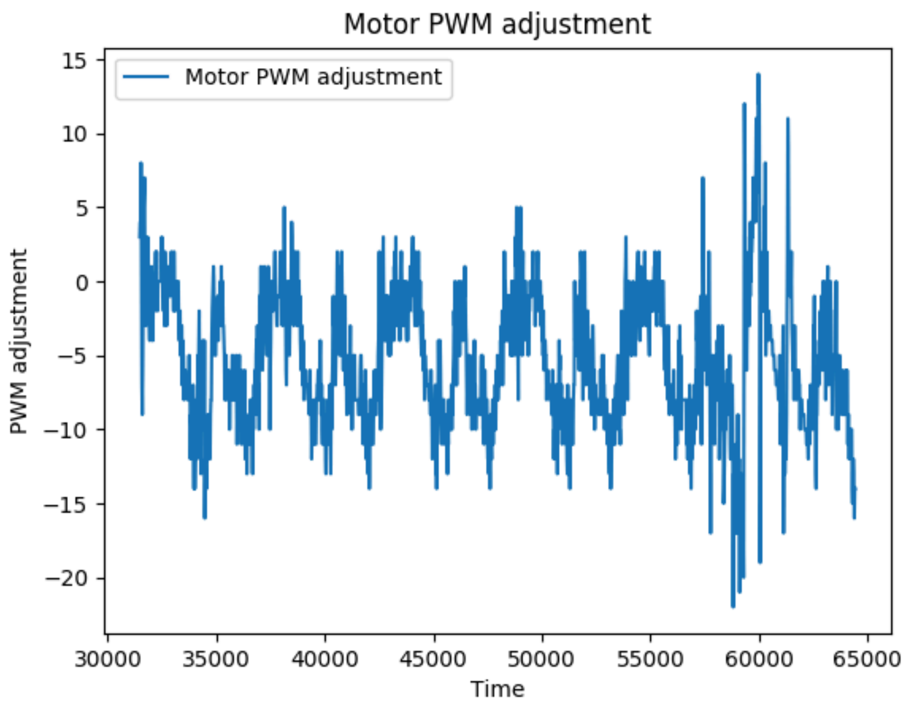
Dominique Blouin, (TPT)

Mechanical engineer

Demo…



Measured error
Measured Error
Error
Time
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
30000
35000
40000
45000
50000
55000
60000
65000



Motor PWM adjustment
Motor PWM adjustment
PWM adjustment
Time
15
10
5
0
−5
−10
−15
−20
30000
35000
40000
45000
50000
55000
60000
65000

Pour aller plus loin

- Syntaxe AADL complète:
  http://perso.telecom-paristech.fr/~borde/aadl/documents/AADL_V2_Syntax_Card.pdf
- Exemples de modèles AADL:
  https://github.com/OpenAADL/AADLib
  https://wiki.sei.cmu.edu/aadl/index.php/Models_examples
- Sites web AADL:
  http://www.aadl.info
  https://wiki.sei.cmu.edu/aadl/index.php/Main_Page

- Outils
  - **OSATE 2:**
    https://wiki.sei.cmu.edu/aadl/index.php/Osate_2
  - **RAMSES:**
    https://mem4csd.telecom-paristech.fr
  - **AADL Inspector:**
    http://www.ellidiss.com/products/aadl-inspector

NOVALINCS
LABORATORY FOR COMPUTER SCIENCE AND INFORMATICS

TELECOM ParisTech
Une école de l'IMT

LTCI
Laboratoire Traitement et Communication de l'Information

Thanks for your attention

<u>Questions ???</u>