Presentation of the AADL: Architecture Analysis and Design Language

Outline

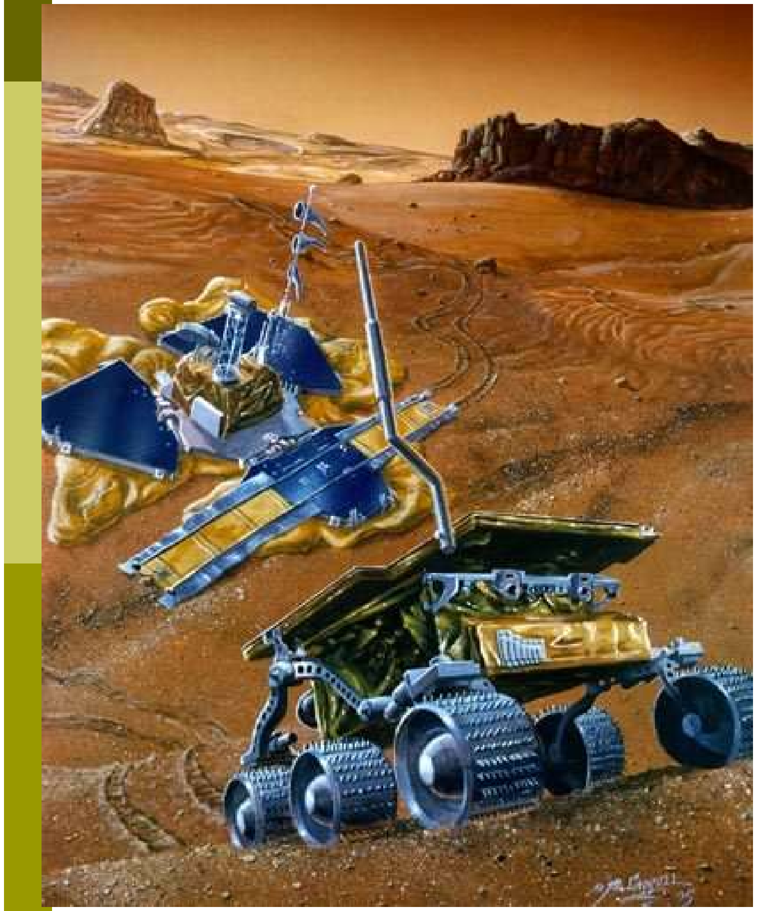
1. **AADL a quick overview**
2. AADL key modeling constructs
   1. AADL components
   2. Properties
   3. Component connection
   4. Behavior annex
3. AADL: tool support

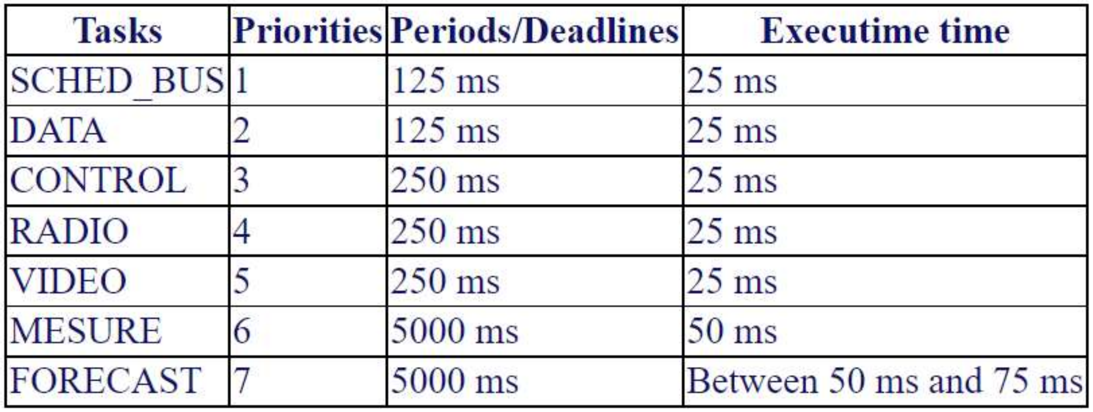
Introduction

- **ADL, Architecture Description Language:**
  - **Goal :** modeling software and hardware architectures to master complexity … to perform analysis
  - **Concepts :** components, connections, deployments.
  - **Many ADLs :** formal/non formal, application domain, …

- **ADL for real-time critical embedded systems:** AADL (*Architecture Analysis and Design Language*).

Example: why ADL and MBSE?

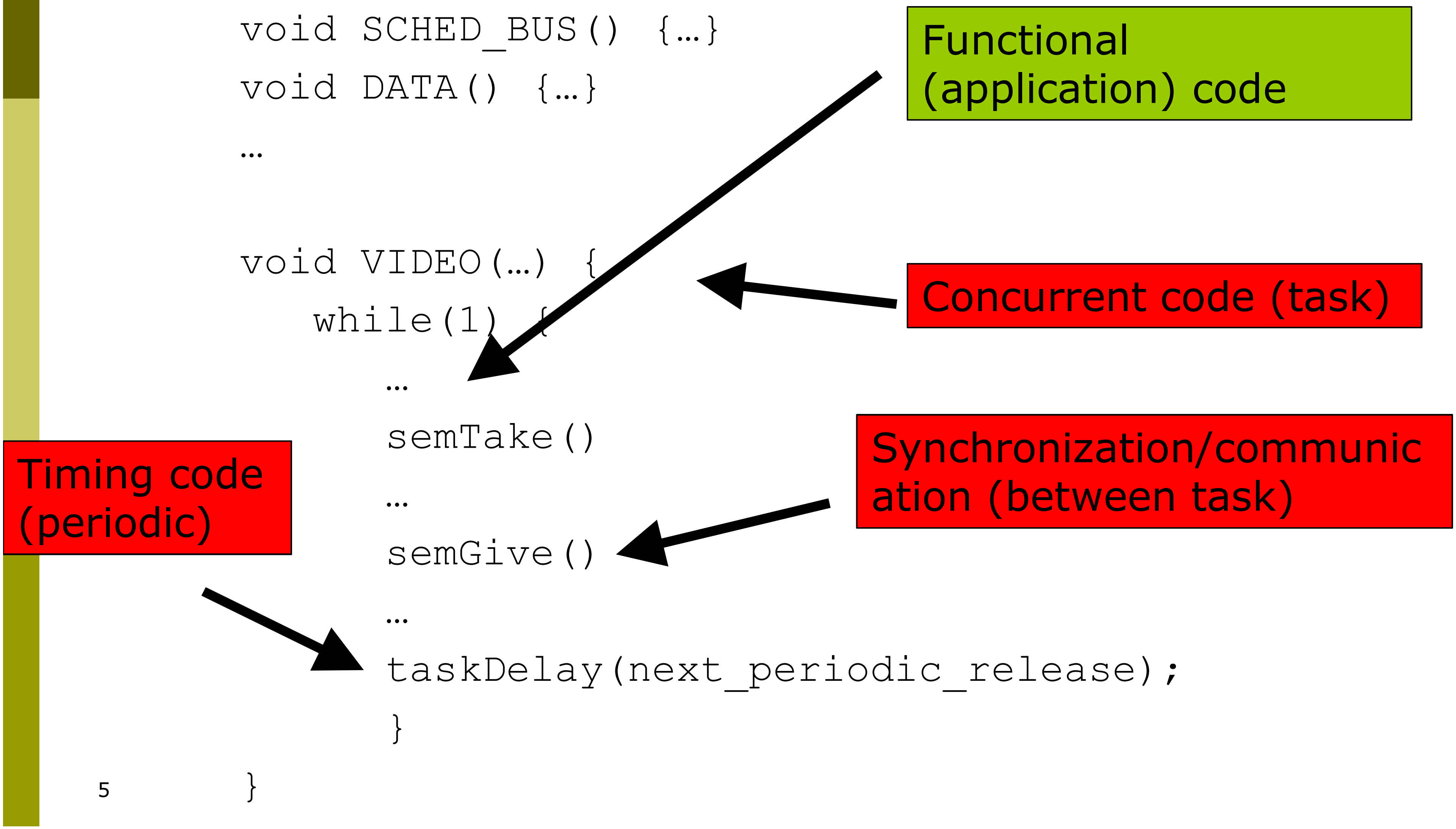




Tasks	Priorities	Periods/Deadlines	Executime time
SCHED_BUS	1	125 ms	25 ms
DATA	2	125 ms	25 ms
CONTROL	3	250 ms	25 ms
RADIO	4	250 ms	25 ms
VIDEO	5	250 ms	25 ms
MESURE	6	5000 ms	50 ms
FORECAST	7	5000 ms	Between 50 ms and 75 ms

- **Mars Pathfinder and its rover Sojourner (1997)**
  - Periodic tasks + synchronization
  - VxWorks operating system
  - Priority inversion bug

Why ADL & MBSE: various concerns in the same real-time program



```
void SCHED_BUS() {…}
void DATA() {…}
…

void VIDEO(…) {
    while(1) {
        …
        semTake()
        …
        semGive()
        …
        taskDelay(next_periodic_release);
        }
}
```

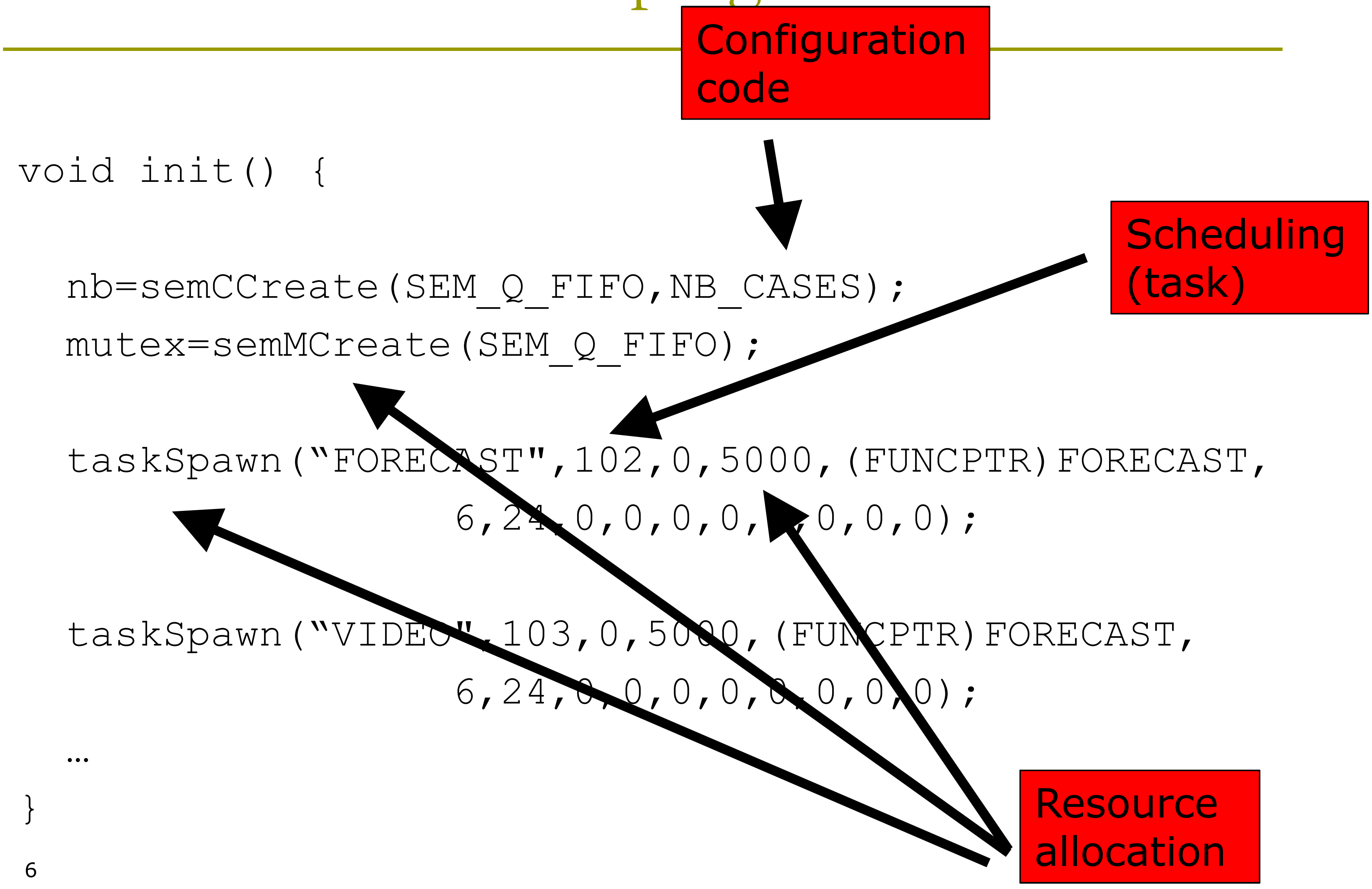
Functional (application) code

Concurrent code (task)

Synchronization/communication (between task)

Timing code (periodic)

Why ADL & MBSE: various concerns/aspects in the same real-time program



Configuration code

Scheduling (task)

Resource allocation

```
void init() {

  nb=semCCreate(SEM_Q_FIFO,NB_CASES);
  mutex=semMCreate(SEM_Q_FIFO);

  taskSpawn(“FORECAST",102,0,5000,(FUNCPTR)FORECAST,
                6,24,0,0,0,0,0,0,0,0);

  taskSpawn(“VIDEO",103,0,5000,(FUNCPTR)FORECAST,
                6,24,0,0,0,0,0,0,0,0);

  …
}
```

Example: why ADL and MBSE?

- **Various concerns/aspects**
  - Functional aspects, but also:
  - Timing aspects (periodic tasks)
  - Concurrency and scheduling (several tasks)
  - Synchronization and communication (between tasks)
  - Resource or operating system configuration

- **Having various concerns make verification, implementation, design space exploration difficult => ADL & MBSE**

AADL: Architecture Analysis & Design Language

- International standard promoted by SAE, AS-2C committee, released as AS5506 family of standards
- Core language document:
  - AADL 1.0 (AS 5506), 2005
  - AADL 2.0 (AS 5506A), 2009
  - Last release: AS 5506D in April 2022
- Annex documents to address specific concerns
  - Annex A: ARINC 653 Interface (AS 5506/1A) 2015
  - Annex B: Data Modelling (AS 5506/2) 2011
  - Annex C: Code Generation Annex (AS 5506/1A) 2015
  - Annex D: Behavior Annex v2 (AS 5506/3) 2017
  - Annex E: Error Model Annex v2 (AS 5506/1A) 2015

AADL is for Analysis

- **AADL objectives are “to model a system”**
  - With analysis in mind (different analysis)
  - To ease transition from well-defined requirements to the final system : code production

- Require semantics => any AADL entity has semantics (natural language or formal methods).

AADL: Architecture Analysis & Design Language

- Different representations :
  - **Textual (standardized representation),**
  - Graphical (declarative and instance views),
  - XML/XMI (not part of the standard: tool specific)

- Graphical editors:
  - OSATE (SEI):
    - declarative model editor
    - instance model viewer
  - MASIW (ISPRAS)
  - Scade Architect (Ansys): instance model editor
  - Stood for AADL (Ellidiss) : instance model editor

AADL components

- **AADL model** : hierarchy/tree of components
  - Composition hierarchy (subcomponents)
  - Inheritance hierarchy (extends)
  - Binding hierarchy (e.g. process->processor)
- **AADL component:**
  - Model a software or a hardware entity
  - May be organized in packages : **reusable**
  - Has a type/interface, zero, one or several implementations
  - May have subcomponents
  - May combine/extend/refine others
  - May have properties : valued typed attributes (source code file name, priority, execution time, memory consumption, …)
- **Component interactions :**
  - Modeled by component connections
  - Binding properties express allocation of SW onto HW

AADL components

- **How to declare a 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 (ex : processor, task, …). Each category has well-defined semantics/behavior, refined through the property and annexes mechanisms
  - Hardware components: execution platform
  - Software components
  - Systems : bounding box of a system. Model deployments.

Component type

- Specification of a component: interface
- All component type declarations follow the same pattern:

<category> foo [**extends** <bar>]

Inherit features and properties from parent

features

  -- *list of features*

  -- *interface*

Interface of the component: Exchange messages, access to data or call subprograms

properties

  -- *list of properties*

  -- *e.g. priority*

Some properties describing non-functional aspect of the component

end foo;

Component type

Example:

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

Standard properties, one can define its own properties

```
                                       -- model a schedulable flow of control
thread bar_thread                      --  bar_thread is a sporadic thread :
features                               --  dispatched whenever it
  in_data : in event data port foo_data; --  receives an event on its  “in_data"
properties                             --  port
  Dispatch_Protocol => Sporadic;
end bar_thread;
```

Component implementation

- Implementation of a component: body
  - Think spec/body package (Ada), interface/class (Java)

```
<category> implementation foo.i [extends <bar>.i]
subcomponents
  …
calls
  -- subprogram subcomponents
  -- called, only for threads or subprograms
connections
properties
  -- list of properties,   e.g. Deadline
end foo.i;
```

`foo.i implements foo`

Component implementation

- **Example:**

```
subprogram Spg
features
  in_param : in parameter foo_data;
properties
   Source_Language => C;
   Source_Text => ("foo.c");
 end Spg;
```

```
thread bar_thread
features
  in_data : in event data port foo_data;
properties
  Dispatch_Protocol => Sporadic;
end bar_thread;
```

Connect data/parameter

```
thread implementation bar_thread.impl        --  in this implementation, at each
calls                                        --  dispatch we execute the "C" call
  C : { S : subprogram spg; };               --  sequence. We pass the dispatch
connections                                  --  parameter to the call sequence
  parameter in_data -> S.in_param;
end bar_thread.impl;
```

AADL concepts

- **AADL introduces many other concepts:**
  - Related to embedded real-time critical systems :
    - AADL flows: capture high-level data+control flows
    - AADL modes: model operational modes in the form of an alternative set of active components/connections/…
  - To ease models design/management:
    - AADL packages (similar to Ada/Java, renames, private/public)
    - AADL abstract component, component extension
    - …
- **AADL is a rich language :**
  - Around 200 entities in the meta-model
  - Around 200 syntax rules in the BNF (core)
  - Around 250 legality rules and more than 500 semantics rules
  - 355 pages core document + various annex documents

Outline

1. AADL a quick overview
2. **AADL key modeling constructs**
    1. **AADL components**
    2. Properties
    3. Component connection
    4. Behavior annex
3. AADL: tool support

AADL workflow

1. Declarative model (Packages)
   - HW libraries
   - SW libraries
   - Applicative composite systems
2. Instance model
   - Selection of the Root System
   - Expanded HW hierarchy
   - Expanded SW hierarchy
3. Deployed model
   - SW instances binding onto HW instances

similar to
UML classes
or SysML blocks

bottom-up

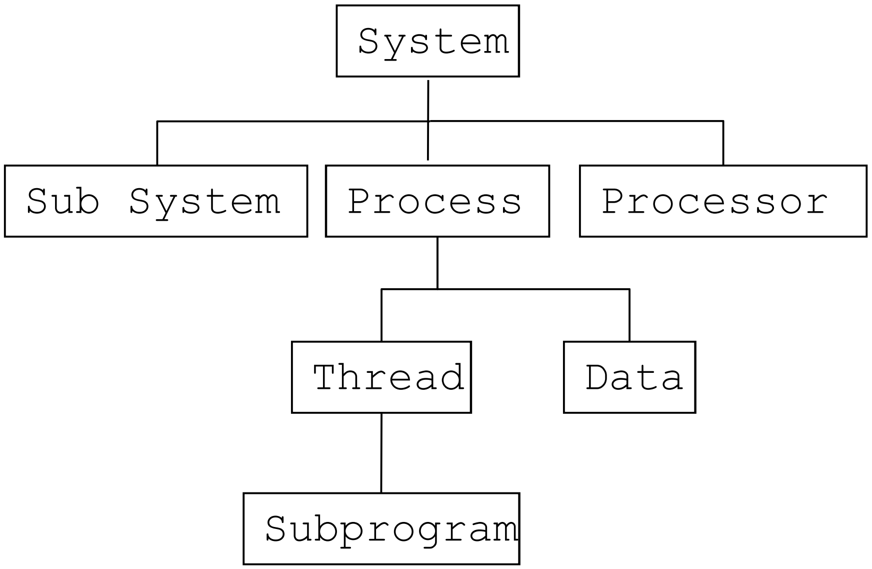
top-down

exhaustive
representation of
the system
hierarchy

required for many
advanced analysis:
-schedulability
-simulation
-safety
-security
-…

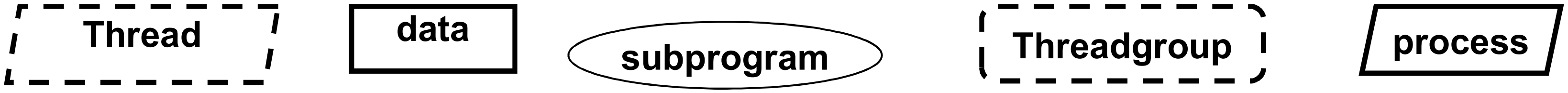
A full AADL system : a tree of component instances

- Component types and implementations only define a library of entities (classifiers)
- An AADL model is a set of component instances (of the classifiers)
- System must be instantiated through a hierarchy of subcomponents, from root (system) to the leafs (subprograms, ..)
- We must choose a system implementation component as the root system model !



Software components 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 :** address space. It must hold at least one thread
- **subprogram :** a sequential execution flow. Associated to a source code (C, Ada) or a model (SCADE, Simulink)
- **thread group :** hierarchy of threads
- **subprogram group** : library or hierarchy of subprograms



Software components

- **Example of a process component :** composed of two threads

```
thread receiver
end receiver;

thread implementation receiver.impl
end receiver.impl;

thread analyser
end analyser;

thread implementation analyser.impl
end analyser.impl;
```

```
process processing
end processing;

process implementation processing.others
subcomponents
   receive : thread receiver.impl;
   analyse : thread analyser.impl;
   . . .
end processing.others;
```

Software components

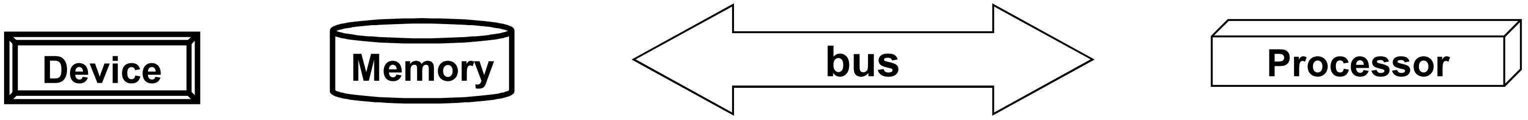
- **Example of a thread component :** a thread may call different subprograms

```
subprogram Receiver_Spg
end Receiver_Spg;

subprogram ComputeCRC_Spg
end ComputeCRC_Spg;

…
```

```
thread receiver
end receiver;

thread implementation  receiver.impl
CS : calls  {
      call1 : subprogram Receiver_Spg;
      call2 : subprogram ComputeCRC_Spg;
        };
end receiver.impl;
```

Hardware components categories

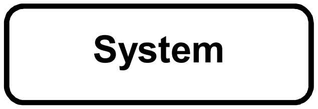
- **processor/virtual processor :** scheduling component **(**combined CPU and OS scheduler).
- **memory :** model data storage (memory, hard drive)
- **device :** component that interacts with the environment. Internals (e.g. firmware) is not modeled.
- **bus/virtual bus :** data exchange mechanism between components



« system » category

- ***system*** **:**

1. Help structuring an architecture, with its own hierarchy of subcomponents. A system can include one or several subsystems.
2. Root system component.
3. Bindings : model the deployment of components inside the component hierarchy.



System

« system » category

```
thread receiver …

thread implementation receiver.impl
Properties
        period => 10 ms;
        dispatch_protocol => periodic;
        deadline => 10 ms;
        priority => 100;
        compute_execution_time =>
                 10 ms .. 20 ms;
end receiver.impl;

process processing
end processing;

process implementation processing.others
subcomponents
  receive : thread receiver.impl;
  analyse : thread analyser.impl;
  . . .
end processing.others;
```

```
processor leon2
  properties
        scheduling_protocol => rm;
end leon2;
```

```
system radar
end radar;

system implementation radar.simple
subcomponents
   main : process processing.others;
   cpu : processor leon2;
properties
   Actual_Processor_Binding =>
      reference cpu applies to main;
end radar.simple;
```

About subcomponents

- Semantics: restrictions apply on subcomponents
  - e.g. hardware cannot contain software, etc

category	allowed subcomponent categories
system	all but thread group and thread
processor	virtual processor, memory, bus
memory	memory, bus
process	thread group, thread, subprogram, data
thread group	thread group, thread, subprogram, data
thread	subprogram, data
subprogram	data
data	data, subprogram

Outline

1. AADL a quick overview
2. **AADL key modeling constructs**
   1. AADL components
   2. **Properties**
   3. Component connection
   4. Behavior annex
3. AADL: tool support

AADL properties

- **Property:**
  - Typed attribute, associated to one or more entities
  - Property definition = name + type + possible owners
  - Property association to a component = property name + value
- Can be propagated to subcomponents: **inherit**
- Can override parent's one, case of extends

- **Allowed types in properties:**
  - **aadlboolean, aadlinteger, aadlreal, aadlstring, range, list, enumeration, record,** user defined (Property type)

AADL properties

- **Property sets :**
  - Group property definitions.
  - Property sets part of the standard, e.g. Thread_Properties.
  - Or user-defined, e.g. for new analysis as power analysis

- **Example :**

```
property set Thread_Properties is
   . . .
   Priority : aadlinteger  applies to (thread, device, …);
   Source_Text : inherit list of aadlstring  applies to (data, port, thread, …);
   . . .
end Thread_Properties;
```

AADL properties

- Properties are typed with units to model physical systems, related to embedded real-time critical systems.

```
property set AADL_Projects is
Time_Units: type units (
   ps,
   ns  => ps  * 1000,
   us  => ns  * 1000,
   ms  => us  * 1000,
   sec => ms  * 1000,
   min => sec * 60,
   hr  => min * 60);
--
end AADL_Projects;
```

```
property set Timing_Properties is

 Time: type aadlinteger
   0 ps .. Max_Time units Time_Units;

 Time_Range: type range of Time;

 Compute_Execution_Time: Time_Range
  applies to  (thread, device, subprogram,
   event port, event data port);

end Timing_Properties;
```

AADL properties

- Properties can apply to (*with increasing priority*)
  - a component type (1)
  - a component implementation (2)
  - a subcomponent (3)
  - a contained element path (4)

```
thread receiver
properties  -- (1)
  Compute_Execution_Time => 3 ms .. 4 ms;
  Deadline => 150 ms ;
end receiver;

thread implementation receiver.impl
properties -- (2)
   Deadline => 160 ms;
end receiver.impl;
```

```
process implementation processing.others
subcomponents
  receive0 : thread receiver.impl;
  receive1 : thread receiver.impl;
  receive2 : thread receiver.impl
       {Deadline => 200 ms;};  -- (3)
properties -- (4)
   Deadline => 300 ms applies to receive1;
end processing.others;
```

Outline

1. AADL a quick overview
2. **AADL key modeling constructs**
    1. AADL components
    2. Properties
    3. **Component connection**
    4. Behavior annex
3. AADL: tool support

Component connection

- **Connection:** model component interactions, control flow and/or data flow. E.g. exchange of messages, access to shared data, remote subprogram call (RPC), …

- **features :** connection point part of the interface. Each *feature* has a name, a direction, and a category

- **Features category:** specification of the type of interaction
  - *event port*: event exchange (e.g. alarm, interrupt)
  - *data port*: data exchange triggered by the scheduler
  - *event data port*: data exchange of data triggered with sender (message)
  - *subprogram parameter*
  - *data access* : access to external data component, possibly shared
  - *subprogram access* : RPC or rendez-vous

- **Features direction for port and parameter:**
  - input (`in`), output (`out`), both (`in out`).

Component connection

- Features of subcomponents are connected in the “connections” subclause of the enclosing component
- Ex: threads & thread connection on data port

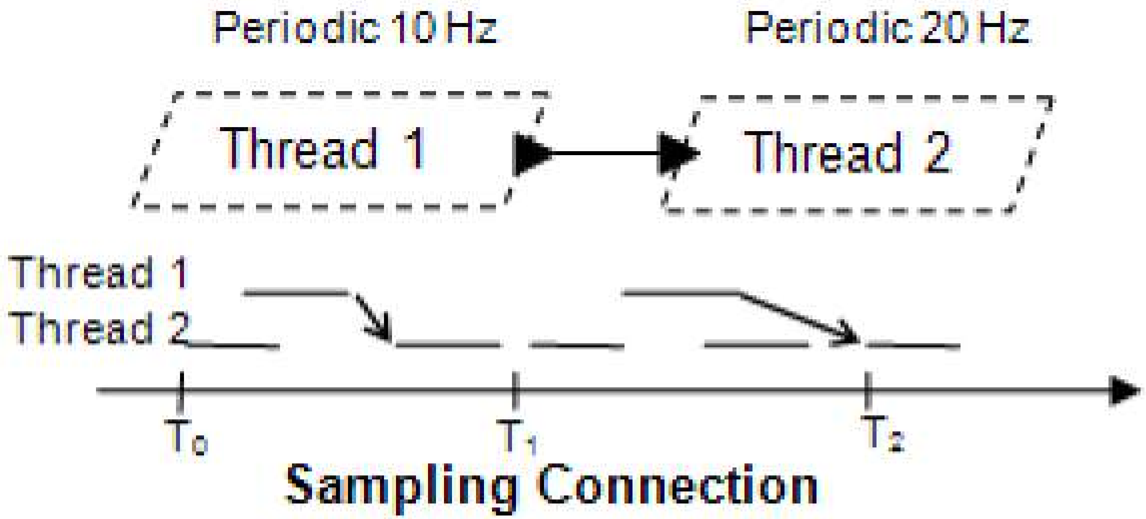
```
thread analyser
features
  analyser_out : out data port
       Target_Position.Impl;
end analyser;

thread display_panel
features
  display_in : in data port Target_Position.Impl;
end display_panel;
```

```
process implementation processing.others
subcomponents
  display : thread display_panel.impl;
  analyse : thread analyser.impl;
connections
  port analyse.analyser_out -> display.display_in;
end processing.others;
```

Data connection policies

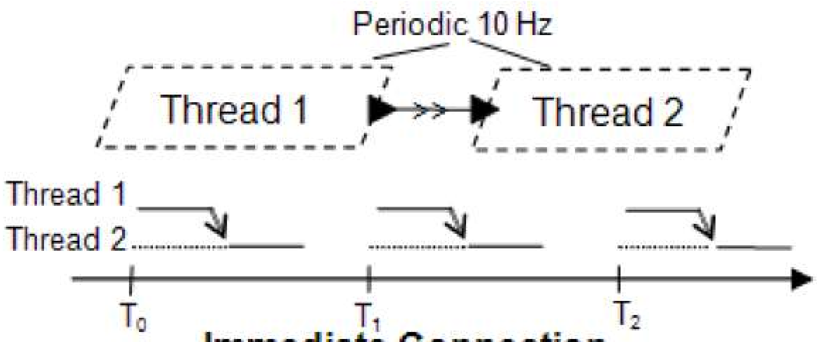
- **Allow predictable communications**
- **Emit at completion time of emitter**
- **Receive at starting time of receiver**
- **Multiple policies exist to control production and consumption of data by threads:**

1. **Sampling connection:** takes the latest value
   - Problem: data consistency (lost or read twice) !

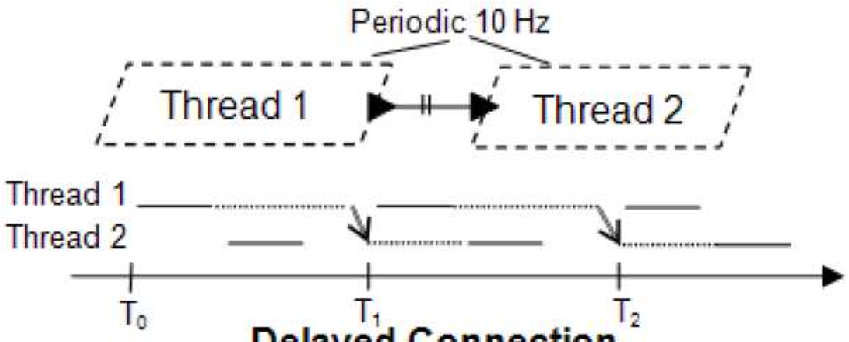


Data connection policies

2. **Immediate:** receiver thread is immediately awaken, and will read data when emitter finished
3. **Delayed:** actual transmission is delayed to the next time frame



Immediate Connection



Delayed Connection

Component connection

❑ Connection for shared data :

```
process implementation processing.others
   subcomponents
      analyse : thread analyser.impl;
      display : thread display_panel.impl;
      a_data  : data shared_var.impl;
   connections
      cx1  : data access a_data -> display.share;
      cx2 : data access a_data -> analyse.share;
end processing.others;

data shared_var
properties
       Concurrency_Control_Protocols
       => PCP;
end shared_var;
```

```
data  implementation shared_var.impl
end shared_var.impl;

thread analyser
features
 share : requires data access shared_var.impl;
end analyser;

thread display_panel
features
  share : requires data access shared_var.impl;
end display_panel;
```

Component connection

- **Connection between *thread* and *subprogram* :**

```
thread implementation  receiver.impl
calls  {
   RS: subprogram Receiver_Spg;
};
connections
  parameter RS.receiver_out -> receiver_out;
  parameter receiver_in -> RS.receiver_in;
end receiver.impl;
```

```
subprogram Receiver_Spg
features
   receiver_out : out parameter
      radar_types::Target_Distance;
   receiver_in : in parameter
      radar_types::Target_Distance;
end Receiver_Spg;

thread receiver
features
  receiver_out : out data port
      radar_types::Target_Distance;
  receiver_in : in data port
      radar_types::Target_Distance;
end receiver;
```

Outline

1. AADL a quick overview
2. **AADL key modeling constructs**
   1. AADL components
   2. Properties
   3. Component connection
   4. **Behavior annex**
3. AADL: tool support

AADL Behavior Annex

- Provides more details on the internal behavior of threads and subprograms.
- Complements, extends or replaces Modes, Calls and some Properties defined in the core model.
- Required for accurate timing analysis and virtual execution of the AADL model.
- State Transition Automata with an action language:
  - dispatch conditions
  - actions: event sending, subprogram call, critical sections, …
  - control structures: loops, tests, …

AADL Behavior Annex example

```
thread transmitter
features
  transmitter_out : out data port radar_types::Radar_Pulse;
end transmitter;

thread implementation transmitter.impl
…
annex Behavior_Specification {**
  states
    s : initial complete final state;
  transisitons
    t : s -[on dispatch]-> s { transmitter_out := "ping" };
**};
end transmitter.impl;
```

annex identifier

state declaration

transition condition

transition actions

Outline

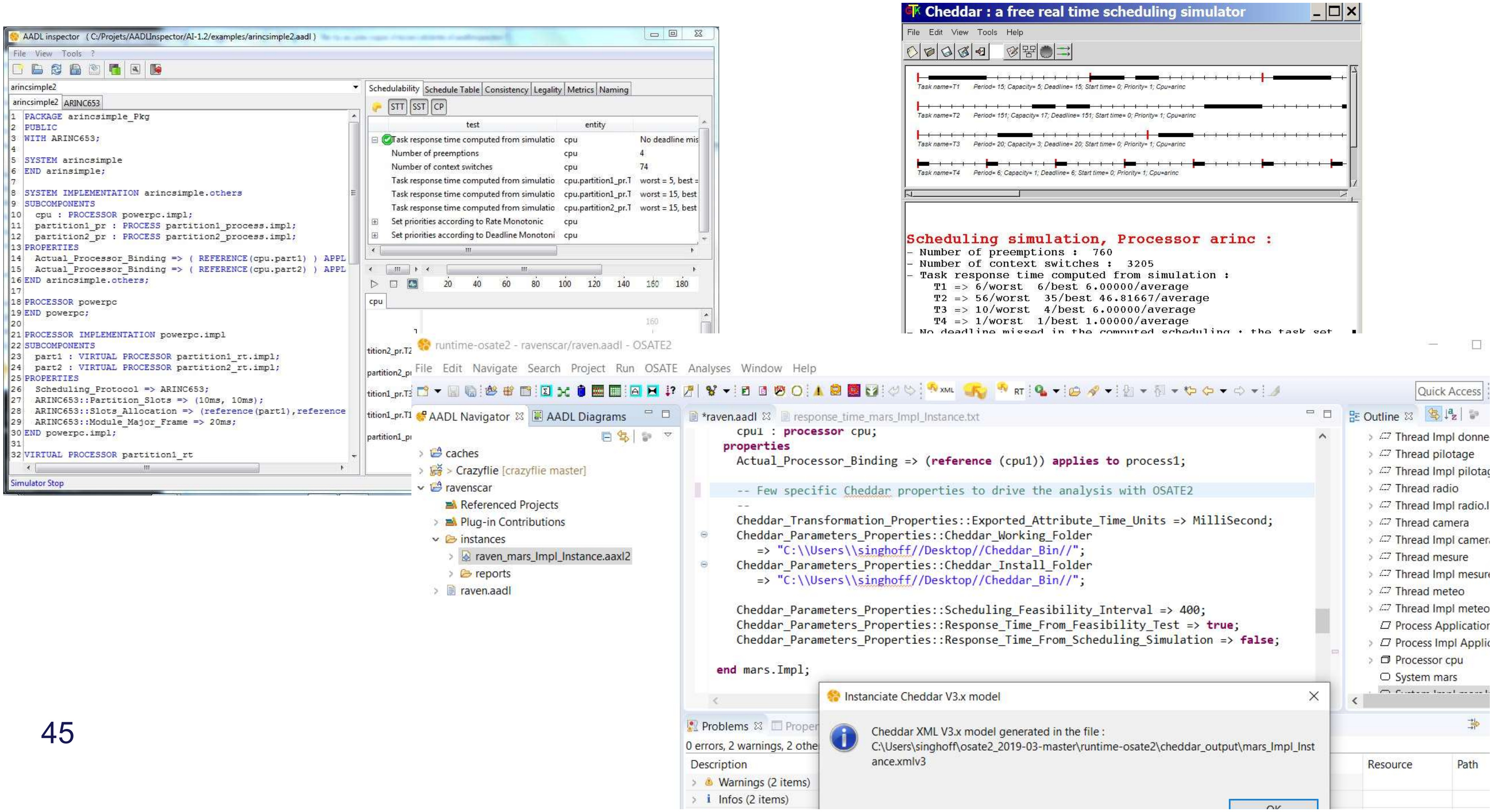
1. AADL a quick overview
2. AADL key modeling constructs
    1. AADL components
    2. Properties
    3. Component connection
    4. Behavior annex
3. **AADL: tool support**

AADL & Tools

- **OSATE** (SEI/CMU, http://osate.org)
  - Eclipse-based tools. Reference implementation.
  - Textual and graphical editors + various analysis plug-ins
- **STOOD** (Ellidiss, http://www.ellidiss.com)
  - Graphical editor, code/documentation generation
  - Guided modeling approach, requirements traceability
- **AADLInspector** (Ellidiss, http://www.ellidiss.com)
  - Standalone framework to process AADL models and Behavior Annex
  - Industrial version of Cheddar + Simulation Engine
- **Ocarina** (ISAE, http://www.openaadl.org)
  - Command line tool, library to manipulate models.
  - AADL parser + code generation + analysis (Petri Net, WCET, …)
- **Cheddar** (UBO/Lab-STICC, http://beru.univ-brest.fr/cheddar/)
  - Performance analysis
- **Others:** RAMSES, PolyChrony, ASSIST, MASIW, MDCF, TASTE, Scade Architect, Camet, Bless, …

Tools used for the tutorial

- AADLInspector, OSATE/Cheddar



AADL inspector (C:/Projets/AADLInspector/AI-1.2/examples/arincsimple2.aadl)

```
PACKAGE arincsimple_Pkg
PUBLIC
WITH ARINC653;

SYSTEM arincsimple
END arinsimple;

SYSTEM IMPLEMENTATION arincsimple.others
SUBCOMPONENTS
  cpu : PROCESSOR powerpc.impl;
  partition1_pr : PROCESS partition1_process.impl;
  partition2_pr : PROCESS partition2_process.impl;
PROPERTIES
  Actual_Processor_Binding => ( REFERENCE(cpu.part1) ) APPL
  Actual_Processor_Binding => ( REFERENCE(cpu.part2) ) APPL
END arincsimple.others;

PROCESSOR powerpc
END powerpc;

PROCESSOR IMPLEMENTATION powerpc.impl
SUBCOMPONENTS
  part1 : VIRTUAL PROCESSOR partition1_rt.impl;
  part2 : VIRTUAL PROCESSOR partition2_rt.impl;
PROPERTIES
  Scheduling_Protocol => ARINC653;
  ARINC653::Partition_Slots => (10ms, 10ms);
  ARINC653::Slots_Allocation => (reference(part1),reference
  ARINC653::Module_Major_Frame => 20ms;
END powerpc.impl;

VIRTUAL PROCESSOR partition1_rt
```

test	entity	
Task response time computed from simulatio	cpu	No deadline mis
Number of preemptions	cpu	4
Number of context switches	cpu	74
Task response time computed from simulatio	cpu.partition1_pr.T	worst = 5, best =
Task response time computed from simulatio	cpu.partition1_pr.T	worst = 15, best
Task response time computed from simulatio	cpu.partition2_pr.T	worst = 15, best
Set priorities according to Rate Monotonic	cpu	
Set priorities according to Deadline Monotoni	cpu	

Cheddar : a free real time scheduling simulator

Task name=T1 Period= 15; Capacity= 5; Deadline= 15; Start time= 0; Priority= 1; Cpu=arinc

Task name=T2 Period= 151; Capacity= 17; Deadline= 151; Start time= 0; Priority= 1; Cpu=arinc

Task name=T3 Period= 20; Capacity= 3; Deadline= 20; Start time= 0; Priority= 1; Cpu=arinc

Task name=T4 Period= 6; Capacity= 1; Deadline= 6; Start time= 0; Priority= 1; Cpu=arinc

```
Scheduling simulation, Processor arinc :
- Number of preemptions :  760
- Number of context switches :  3205
- Task response time computed from simulation :
    T1 => 6/worst  6/best 6.00000/average
    T2 => 56/worst  35/best 46.81667/average
    T3 => 10/worst  4/best 6.00000/average
    T4 => 1/worst  1/best 1.00000/average
- No deadline missed in the computed scheduling : the task set
```

runtime-osate2 - ravenscar/raven.aadl - OSATE2

File Edit Navigate Search Project Run OSATE Analyses Window Help
XML
RT
Quick Access
AADL Navigator
AADL Diagrams
caches
> Crazyflie [crazyflie master]
ravenscar
Referenced Projects
Plug-in Contributions
instances
raven_mars_Impl_Instance.aaxl2
reports
raven.aadl
*raven.aadl
response_time_mars_Impl_Instance.txt

```
    cpu1 : processor cpu;
  properties
    Actual_Processor_Binding => (reference (cpu1)) applies to process1;

    -- Few specific Cheddar properties to drive the analysis with OSATE2
    --
    Cheddar_Transformation_Properties::Exported_Attribute_Time_Units => MilliSecond;
    Cheddar_Parameters_Properties::Cheddar_Working_Folder
      => "C:\\Users\\singhoff//Desktop//Cheddar_Bin//";
    Cheddar_Parameters_Properties::Cheddar_Install_Folder
      => "C:\\Users\\singhoff//Desktop//Cheddar_Bin//";

    Cheddar_Parameters_Properties::Scheduling_Feasibility_Interval => 400;
    Cheddar_Parameters_Properties::Response_Time_From_Feasibility_Test => true;
    Cheddar_Parameters_Properties::Response_Time_From_Scheduling_Simulation => false;

end mars.Impl;
```

0 errors, 2 warnings, 2 othe

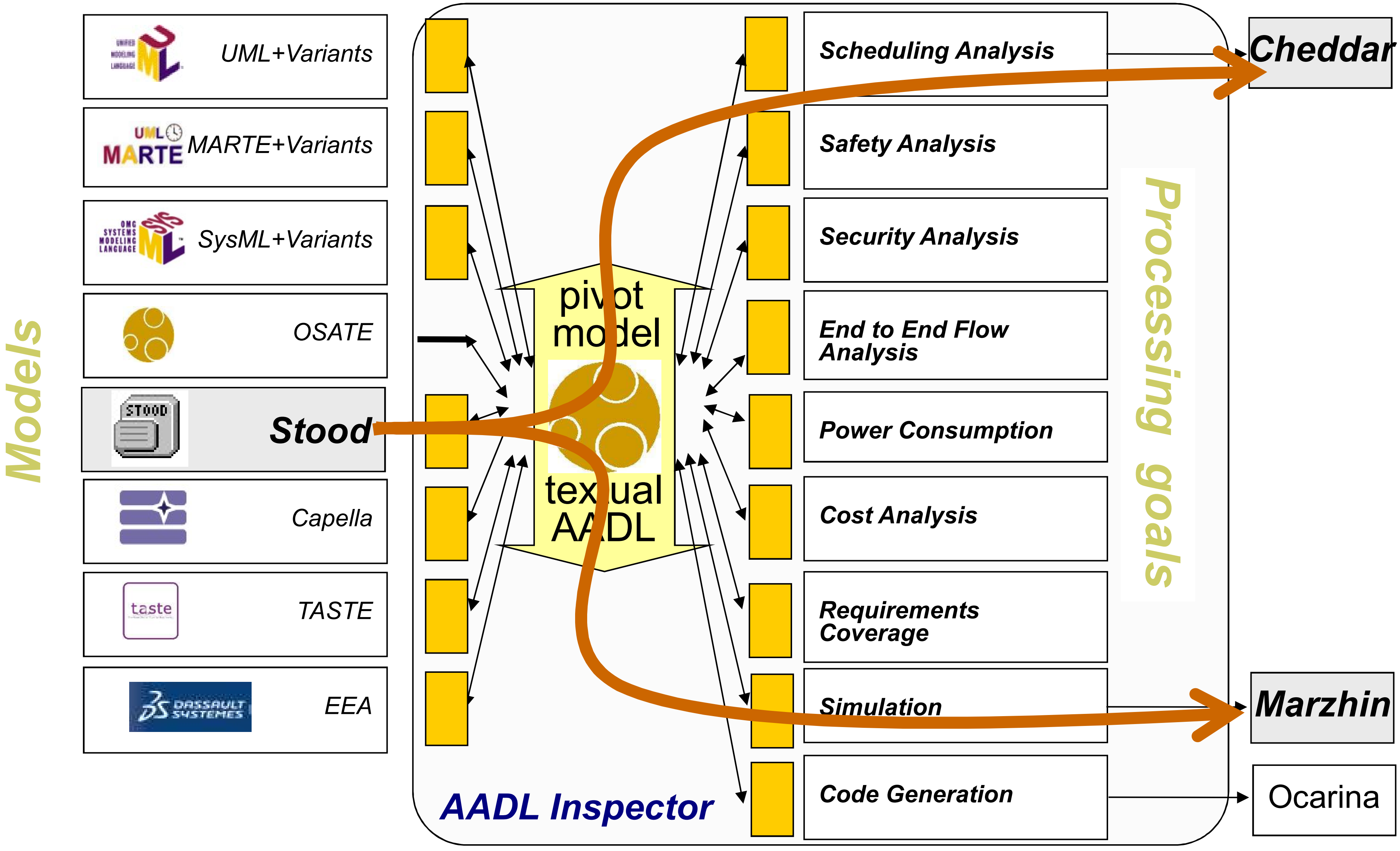
Warnings (2 items)

Infos (2 items)

Instanciate Cheddar V3.x model

Cheddar XML V3.x model generated in the file :
C:\Users\singhoff\osate2_2019-03-master\runtime-osate2\cheddar_output\mars_Impl_Instance.xmlv3

Tools used for the tutorial



Models
UML+Variants
MARTE+Variants
SysML+Variants
OSATE
Stood
Capella
TASTE
EEA
pivot model
textual AADL
AADL Inspector
Scheduling Analysis
Safety Analysis
Security Analysis
End to End Flow Analysis
Power Consumption
Cost Analysis
Requirements Coverage
Simulation
Code Generation
Processing goals
Cheddar
Marzhin
Ocarina