Scheduling analysis of AADL architecture models

Frank Singhoff+, Pierre Dissaux*

+Lab-STICC/CNRS UMR 6285, Université de Bretagne Occidentale, France
***Ellidiss Technologies, France**


Outline

Goal: overview of scheduling analysis capabilities that are proposed by the AADL and tools implementing it. Show the benefits that can be expected by performing early scheduling analysis for real-time software.

- **Part 1: Introduction to AADLv2 core (about 100')**
  - Syntax, semantics of the language
- **Part 2: introducing a case study (about 20')**
  - A radar illustrative case study
- **Part 3: Scheduling analysis (about 90')**
  - Introducing real-time scheduling and its use with AADL
- **Part 4 : practical labs, exercises (about 2/3 hours)**
  - How to apply what we learnt in parts 1 to 3

2

Acknowledgments

- Many of those slides were written with or by Jérôme Hugues/ISAE, for the following tutorials:
  - AADLv2, An Architecture Description Language for the Analysis and Generation of Embedded Systems. J. Hugues, F. Singhoff. Half day tutorial presented in the ACM HILT conference, Portland, USA, October 2014.
  - AADLv2, a Domain Specific Language for the Modeling, the Analysis and the Generation of Real-Time Embedded Systems. F. Singhoff, J. Hugues. Half day tutorial presented in the International MODELS conferences, Valencia, Spain, September 2014.
  - AADLv2, an Architecture Description Language for the Analysis and Generation of Embedded Systems. J. Hugues F. Singhoff. Half day tutorial presented in the International EMSOFT/ESWEEK conferences, Montreal, Canada, September 2013.
  - Développement de systèmes à l'aide d'AADL - Ocarina/Cheddar. J. Hugues, F. Singhoff. Tutoriel présenté à l'école d'été temps réel (ETR'2009). Septembre 2009. Pages 25-34. Paris.
- Thank you Jérôme :-)

3

We focus on Real-Time, Critical, Embedded Systems

- *« The correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced »* Stankovic, 1988.
- **Properties we look for:**
  - Functions must be predictable: the same data input will produce the same data output.
  - Timing behavior must be predictable: must meet temporal constraints (e.g. deadline).

4

We focus on Real-Time, Critical, Embedded Systems

- **Critical real-time systems:** temporal constraints MUST be met, otherwise defects could have a dramatic impact on human life, on the environment, on the system,
- **Embedded systems:** computing system designed for specific control functions within a larger system.
  - Often with temporal constraints.
  - Part of a complete device, often including hardware and mechanical parts
  - Limited amount of resources.

5

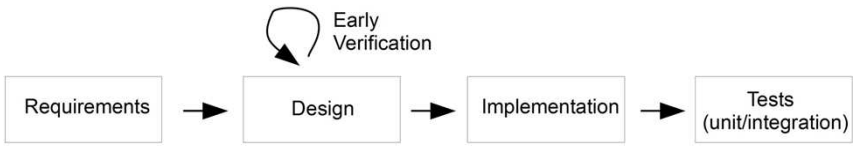
We focus on Real-Time, Critical, Embedded Systems

- **Examples:** aircraft, satellite, automotive, …

1. Need to handle time. Concurrent applications.
2. May have dramatic impact on human life, on the system, ...
3. Do not allow software maintenance => difficult to correct erroneous software/bugs.
4. High implementation cost : temporal constraints verification, safety, dedicated hardware/software

6

We focus on Real-Time, Critical, Embedded Systems



- **Specific software engineering** methods/models/tools to master quality and cost
  - Example : early verifications at design step

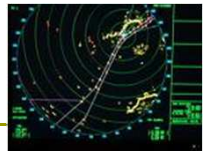
7

Motivation for early verification

- **From NIST 2012:**
  - 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, e.g. can deadlines be met?

8

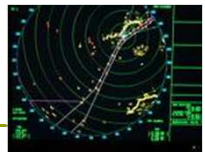
Objectives of this tutorial



- Issues
  - How to model/design a real-time critical embedded system that conforms to requirements?
  - How to verify the solution?
  - How to simulate it?
- One solution amoung others: use an architecture description language
  - to model the system,
  - to run various verification,
  - and to automatically produce the system
- Focus on the AADL2.2 SAE standard

9

Objectives of this tutorial



- Illustration: model of a simple radar system
- Let us suppose we have the following requirements

1. System implementation is composed by physical devices (Hardware entity): antenna + processor + memory + bus
2. and software entities : running processes and threads + operating system functionalities (scheduling) implemented in the processor that represent a part of execution platform and physical devices in the same time.
3. The main process is responsible for signals processing : general pattern: transmitter -> antenna -> receiver -> analyzer -> display
4. Analyzer is a periodic thread that compares transmitted and received signals to perform detection, localization and identification.
5. [..]

10

Outline

Goal: overview of scheduling analysis capabilities that are proposed by the AADL and tools implementing it. Show the benefits that can be expected by performing early scheduling analysis for real-time software.

- **Part 1: Introduction to AADLv2 core (about 100')**
  - Syntax, semantics of the language
- **Part 2: introducing a case study (about 20')**
  - A radar illustrative case study
- **Part 3: Scheduling analysis (about 90')**
  - Introducing real-time scheduling and its use with AADL
- **Part 4 : practical labs, exercises (about 2/3 hours)**
  - How to apply what we learnt in parts 1 to 3

11

Resources for this tutorial

- Information on AADL
  - http://www.aadl.info : updates on AADL standard
  - http://www.openaadl.org : other AADL resources
  - http://beru.univ-brest.fr/~singhoff/cheddar/: Cheddar and real-time scheduling
  - http://www.ellidiss.fr/: AADLInspector and Ellidiss Tech. AADL activities
- Feel free to contact us for more details

12

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*).

3

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) 2004
  - AADL 2.0 (AS 5506A) 2009
  - AADL 2.1 (AS 5506B) 2012
  - AADL 2.2 (AS 5506C) 2017
- 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

4

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).

5

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 only
  - MASIW (ISPRAS)
  - Scade Architect (Ansys): instance model editor
  - Stood for AADL (Ellidiss) : instance model editor

6

AADL components

- **AADL model** : hierarchy/tree of components
  - Composition hierarchy (subcomponents)
  - Inheritance hierarchy (extends)
  - Binding hierarchy (e.g. process->virtual processor->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

7

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.

8

Component type

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

```
<category> foo [extends <bar>]
features
  -- list of features
  -- interface
properties
  -- list of properties
  --  e.g. priority
end foo;
```

Inherit features and properties from parent

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

Some properties describing non-functional aspect of the component

Component type

- **Example:**

```
                                                    -- 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;


                                                                -- schedula
control flow
  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" port
  properties
    Dispatch_Protocol => Sporadic;
```

Standard properties, one can define its own properties

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                                    --  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;

  thread bar_thread                                    --
bar_thread is a sporadic thread,
  features                                                -
it is 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;

  thread implementation bar_thread.impl    --  in this
implementation, at each
  calls
```

Connect data/parameter

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

13

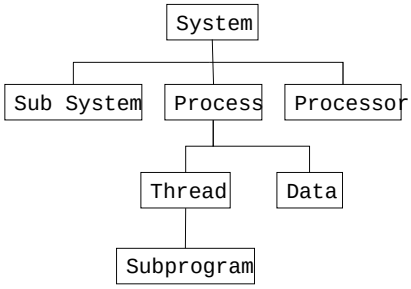
Outline

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

14

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 !



15

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

bottom-up

top-down

similar to UML classes or SysML blocks

exhaustive representation of the system hierarchy

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

16

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

17

Software components

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

```
process processing
end processing;

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

```
thread receiver
end receiver;

thread implementation
receiver.impl
end receiver.impl;

thread analyser
end analyser;

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

18

Software components

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

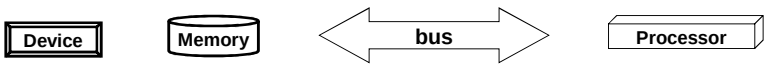
```
thread receiver
end receiver;

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

```
subprogram Receiver_Spg
end Receiver_Spg;

subprogram ComputeCRC_Spg
end ComputeCRC_Spg;
. . .
```

19

Hardware components categories

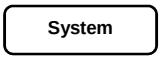
- **processor/virtual processor :** scheduling component **(**combined CPU and OS scheduler).
- **memory :** model data storage (memory, hard drive)
- **device :** component that interacts with the environment. Internals (e.g. firmware) is not modeled.
- **bus/virtual bus :** data exchange mechanism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



20

« 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 » category

```
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;

processor leon2
end leon2;
```

```
device antenna
end antenna;
```

```
subprogram Receiver_Spg …
thread receiver …

thread implementation
receiver.impl
… call1 : subprobram
Receiver_Spg; …
end receiver.impl;

process processing
end processing;

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

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

23

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

24

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)

25

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;
```

26

AADL properties

- Properties may be typed with units to model physical systems, related to embedded real-time critical systems.
- Examples: Time_Units, Size_Units, Data_Rate_Units, Processor_Speed_Units

```
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 appy 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
.. 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
```

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

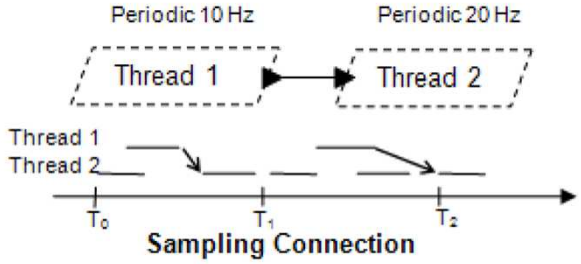
- Features of subcomponents are connected in the “connections” subclause of the enclosing component
- Ex: threads & thread connection on data port

```
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;
```

```
thread analyser
features
   analyser_out : out
data port

Target_Position.Impl;
end analyser;

thread display_panel
features
   display_in : in data
port
```

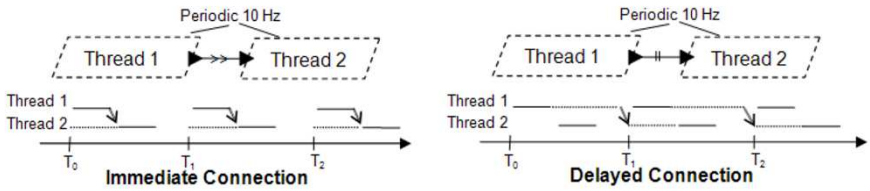

Data connection policies

- **Allow deterministic communications**
- **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



33

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
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;
```

34

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;
```

35

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

36

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, …

37

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;
  TRANSITIONS
    t : s -[ON DISPATCH]-> s { transmitter_out :=
"ping" };
**};
END transmitter.impl;
```

annex identifier

state declaration

transition condition

transition actions

38

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**

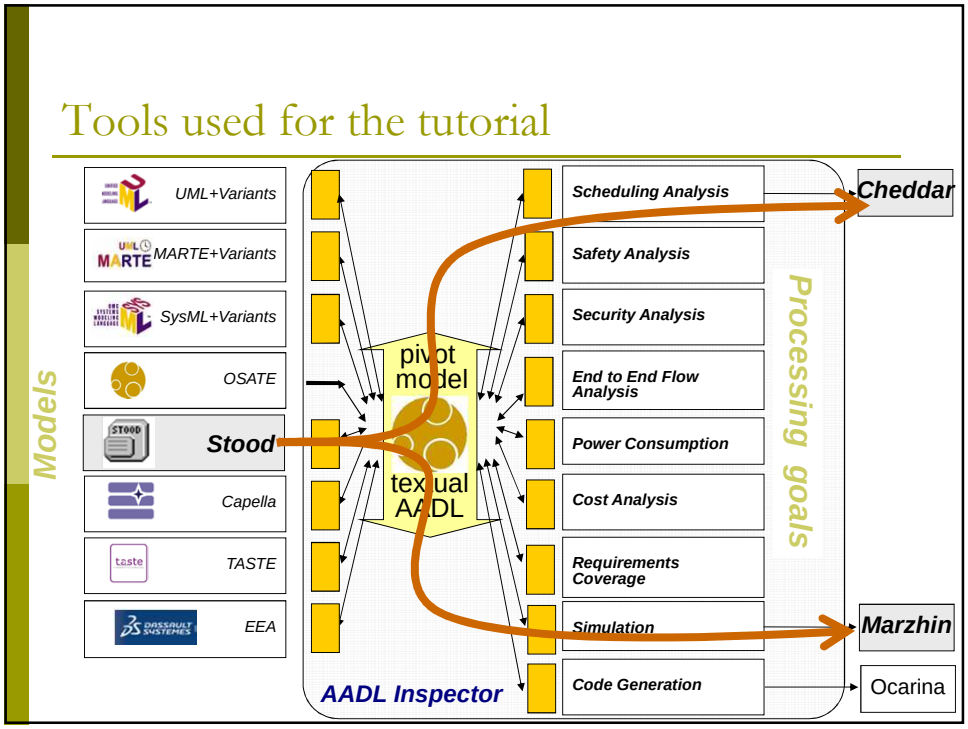
39

AADL & Tools

- **OSATE** (SEI/CMU, http://aadl.info)
  - 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
- **Cheddar** (UBO/Lab-STICC, http://beru.univ-brest.fr/~singhoff/cheddar/)
  - Performance analysis
- **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, …)
- **Others:** RAMSES, PolyChrony, ASSIST, MASIW, MDCF, TASTE, Scade Architect, Camet, Bless

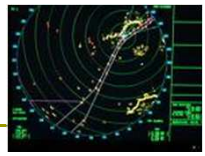
40

Tools used for the tutorial



AADL : a radar case study

Back to radar case study



- Goal: to model a simple radar system
- Let us suppose we have the following requirements

1. System implementation is composed by physical devices (Hardware entity): antenna + processor + memory + bus
2. and software entity : running processes and threads + operating system functionalities (scheduling) implemented in the processor that represent a part of execution platform and physical devices in the same time.
3. The main process is responsible for signals processing : general pattern: transmitter -> antenna -> receiver -> analyzer -> display
4. Analyzer is a periodic thread that compares transmitted and received signals to perform detection, localization and identification.
5. [..]

2

Tools used for modeling

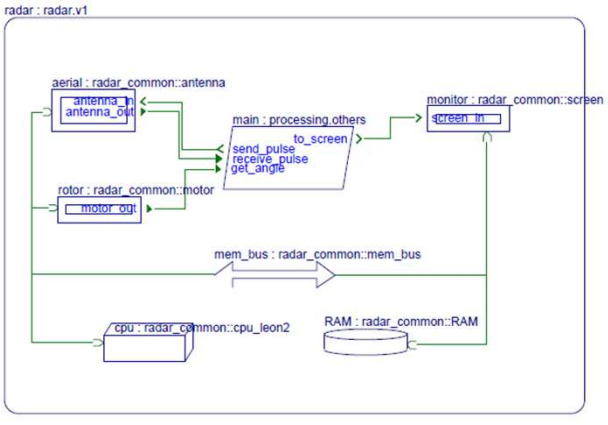
- AADL syntax is both textual and graphical, with several editors available
  - Modes exist for emacs, vi
  - OSATE provides a comprehensive textual IDE on top of Eclipse, and additional plug-ins
    - IMV : Instance Model Viewer
    - Consistency checkers, statistics, etc.
  - Stood for AADL:
    - Top-down modeling approach
    - Instance Model graphical editor
    - Generation of textual AADL for tool interoperability
- In the following, we will use Stood

3

Radar case study

- Hardware/Software breakdown: components

```
PACKAGE radar_v1
PUBLIC
-- …
SYSTEM radar
END radar;
-- …
PROCESS processing
-- …
END processing;
-- …
END radar_v1;

PACKAGE radar_common
PUBLIC
-- …
DEVICE screen
-- …
END screen;
-- …
END radar_common;
```

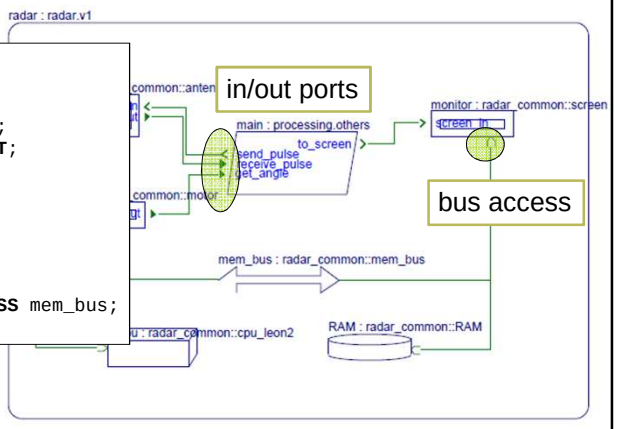


4

Radar case study

- Hardware/Software breakdown: features

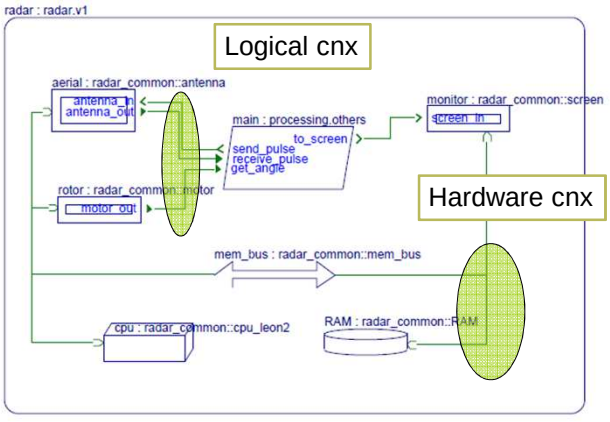
```
PROCESS processing
FEATURES
  to_screen : OUT EVENT PORT;
  send_pulse : OUT EVENT PORT;
  receive_pulse : IN DATA PORT;
  get_angle : IN DATA PORT;
END processing;

DEVICE screen
FEATURES
  screen_in : IN EVENT PORT;
  mem_bus : REQUIRES BUS ACCESS mem_bus;
END screen;
```



5

Radar case study

- Hardware/Software breakdown: connections



6

Radar case study

- Hardware/Software breakdown: connections

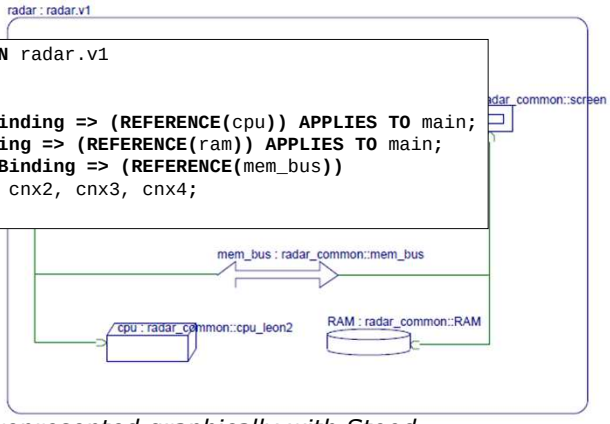
```
SYSTEM IMPLEMENTATION radar.v1
SUBCOMPONENTS
  aerial : DEVICE radar_common::antenna;
  rotor : DEVICE radar_common::motor;
  monitor : DEVICE radar_common::screen;
  cpu : PROCESSOR radar_common::cpu_leon2;
  mem_bus : BUS radar_common::mem_bus;
  RAM : MEMORY radar_common::RAM;
  main : PROCESS processing.others;
CONNECTIONS
  cnx1 : PORT aerial.antenna_out -> main.receive_pulse;
  cnx2 : PORT rotor.motor_out -> main.get_angle;
  cnx3 : PORT main.send_pulse -> aerial.antenna_in;
  cnx4 : PORT main.to_screen -> monitor.screen_in;
  cnx5 : BUS ACCESS mem_bus -> aerial.mem_bus;
  cnx6 : BUS ACCESS mem_bus -> rotor.mem_bus;
  cnx7 : BUS ACCESS mem_bus -> monitor.mem_bus;
  cnx8 : BUS ACCESS mem_bus -> cpu.mem_bus;
  cnx9 : BUS ACCESS mem_bus -> RAM.mem_bus;
-- …
END radar.v1;
```

7

Radar case study

- Hardware/Software breakdown: bindings

```
SYSTEM IMPLEMENTATION radar.v1
-- …
PROPERTIES
  Actual_Processor_Binding => (REFERENCE(cpu)) APPLIES TO main;
  Actual_Memory_Binding => (REFERENCE(ram)) APPLIES TO main;
  Actual_Connection_Binding => (REFERENCE(mem_bus))
    APPLIES TO cnx1, cnx2, cnx3, cnx4;
END radar.v1;
```

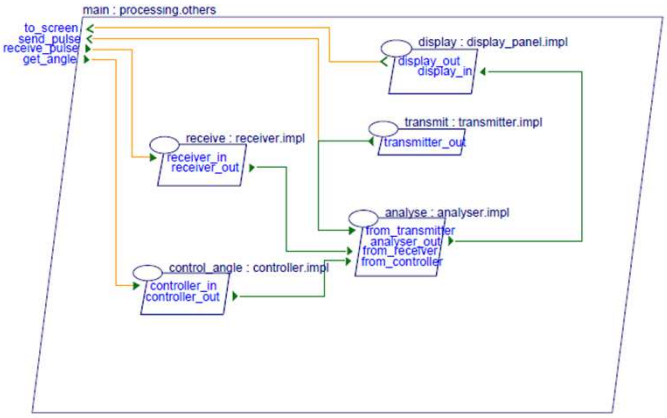


note:
bindings are not represented graphically with Stood

8

Radar case study

- Software elements



9

A few words on AADL usage

- AADL is for architectural description and early analysis
- Not to be compared with UML suites
  - Not a graphical representation of the source code
  - But can be associated with existing source code via Properties
- Keep in mind models support an objective
  - For now, it is just a high-level view of the design
- In the next sections, we will complete the models with
  - Properties to support schedulability analysis
  - Elements to perform virtual execution

10

AADL : about scheduling analysis

Scheduling analysis, what is it ?

- **Embedded real-time critical systems** have temporal constraints to meet (e.g. deadline).
- Many systems are built with operating systems providing multitasking facilities … Tasks may have deadline.
- **But, tasks make temporal constraints analysis difficult to do :**
  - We must take the task scheduling into account in order to check task temporal constraints.
  - Scheduling (or schedulability) analysis.

Summary

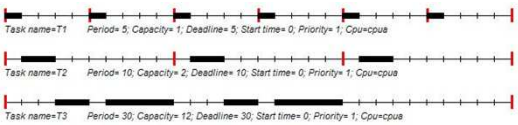
1. Issues about real-time scheduling analysis : AADL to the rescue
2. Basics on scheduling analysis : fixed-priority scheduling for uniprocessor architectures
3. AADL components/properties to scheduling analysis

Real-Time scheduling theory

1. **A set of simplified tasks models** (to model functions of the system)
2. **A set of analytical methods** (called feasibility tests)
   - **Example:**

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

3. **A set of scheduling algorithms**: build the full scheduling/GANTT diagram



Real-Time scheduling theory is hard to apply

- Real-Time scheduling theory
  - Theoretical results defined from 1974 to 1994: feasibility tests exist for uniprocessor architectures
- Now supported at a decent level by POSIX 1003 real-time operating systems, ARINC653, …
- Industry demanding
  - Yet, hard to use

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

Real-Time scheduling theory is hard to apply

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

It is the role of an ADL to hide those details

AADL to the rescue?

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

AADL to the rescue?

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

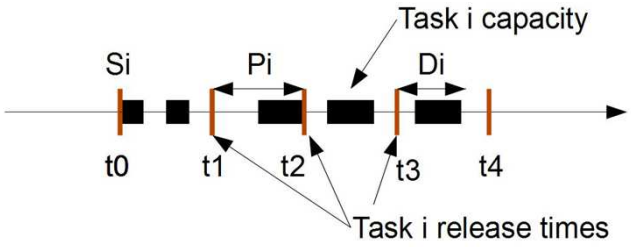
Summary

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

Real-time scheduling theory : models of task

- **Task simplified model:** sequence of statements + data.
- **Usual kind of tasks:**
  - Independent tasks or dependent tasks.
  - Periodic and sporadic tasks (critical functions) : have several jobs and release times
  - Aperiodic tasks (non critical functions) : only one job and one release time

Real-time scheduling theory : models of task



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

Real-time scheduling theory : models of task

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

Uniprocessor fixed priority scheduling

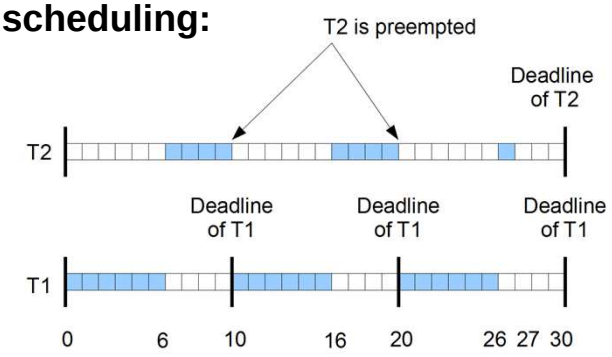
- **Fixed priority scheduling :**
  - Scheduling based on fixed priority => priorities do not change during execution time.
  - Priorities are assigned at design time (off-line).
  - Efficient and simple feasibility tests.
  - Scheduler easy to implement into real-time operating systems.
- **Priority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

Uniprocessor fixed priority scheduling

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

Uniprocessor fixed priority scheduling

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



  - Assuming VxWorks priority levels (high=0 ; low=255)
  - T1 : C1=6, P1=10, Prio1=0
  - T2 : C2=9, P2=30, Prio2=1

Uniprocessor fixed priority scheduling

- **Feasibility/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 factor test:**
     $U = \sum_{i=1}^{n} Ci/Pi \leq n.(2^{\frac{1}{n}}\text{-}1)$ (about 69%)
     Rate Monotonic assignment and preemptive scheduling. Sufficient but not necessary condition.
  3. **Task worst case response time, noted Ri** : delay between task release time and task completion time. Any priority assignment but preemptive scheduling.

Uniprocessor fixed priority scheduling

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

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

  - hp(i) is the set of tasks which have a higher priority than task i.
  - $\lceil x \rceil$ returns the smallest integer not smaller than x.

Uniprocessor fixed priority scheduling

- To compute task response time: compute wi^k with:

$$wi^n = Ci + \sum_{j \in hp(i)} \lceil wi^{n-1}/Pj \rceil . Cj$$

- Start with wi^0=Ci.
- Compute wi^1, wi^2, wi^3, ... wi^k upto:
  - If wi^k >Pi. No task response time can be computed for task i. Deadlines will be missed !
  - If $wi^k = wi^{k-1}$. wi^k is the task i response time. Deadlines will be met.

Uniprocessor fixed priority scheduling

- **Example:** T1(P1=7, C1=3), T2 (P2=12, C2=2), T3 (P3=20, C3=5)

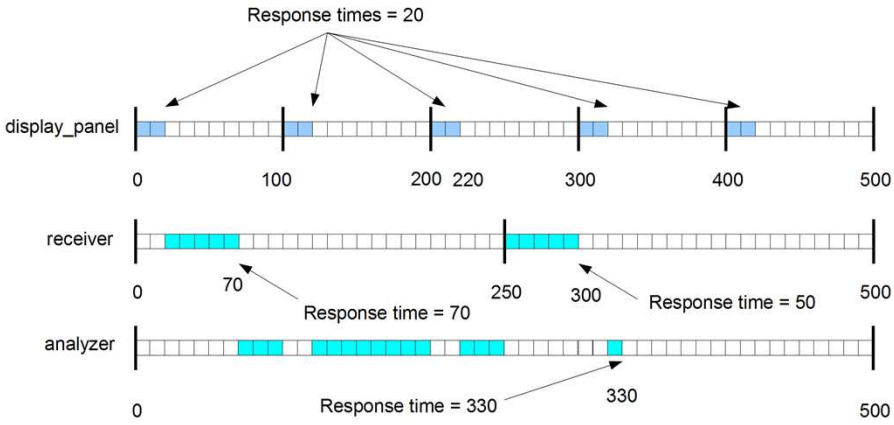
$w1^0 = C1 = 3 \Rightarrow R1 = 3$
$w2^0 = C2 = 2$
$w2^1 = C2 + \left\lceil \frac{w2^0}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{2}{7} \right\rceil . 3 = 5$
$w2^2 = C2 + \left\lceil \frac{w2^1}{P1} \right\rceil . C1 = 2 + \left\lceil \frac{5}{7} \right\rceil . 3 = 5 \Rightarrow R2 = 5$

$w3^0 = C3 = 5$
$w3^1 = C3 + \left\lceil \frac{w3^0}{P1} \right\rceil . C1 + \left\lceil \frac{w3^0}{P2} \right\rceil . C2 = 10$
$w3^2 = C3 + \left\lceil \frac{w3^1}{P1} \right\rceil . C1 + \left\lceil \frac{w3^1}{P2} \right\rceil . C2 = 13$
$w3^3 = C3 + \left\lceil \frac{w3^2}{P1} \right\rceil . C1 + \left\lceil \frac{w3^2}{P2} \right\rceil . C2 = 15$
$w3^4 = C3 + \left\lceil \frac{w3^3}{P1} \right\rceil . C1 + \left\lceil \frac{w3^3}{P2} \right\rceil . C2 = 18$
$w3^5 = C3 + \left\lceil \frac{w3^4}{P1} \right\rceil . C1 + \left\lceil \frac{w3^4}{P2} \right\rceil . C2 = 18 \Rightarrow R3 = 18$

Uniprocessor fixed priority scheduling

- **Example with the AADL case study:**
  - "display_panel" thread which displays data. P=100, C=20.
  - "receiver" thread which sends data. P=250, C=50.
  - "analyser" thread which analyzes data. P=500, C=150.
- **Processor utilization factor test:**
  - U=20/100+150/500+50/250=0.7
  - Bound=3.$(2^{\frac{1}{3}}-1)$=0.779
  - U≤Bound => deadlines will be met.
- **Task response time:** $R_{analyser}$=330, $R_{display\_panel}$=20, $R_{receiver}$ =70.
- **Run simulations on feasibility interval:** [0,LCM(Pi)] = [0,500].

Uniprocessor fixed priority scheduling



Fixed priority and shared resources

- Previous tasks were independent … does not really exist in real life.

- **Task dependencies :**
  - Shared resources.
    - E.g. with AADL: threads may wait for AADL protected data component access.
  - Precedencies between tasks.
    - E.g with AADL: threads exchange data by data port connections.

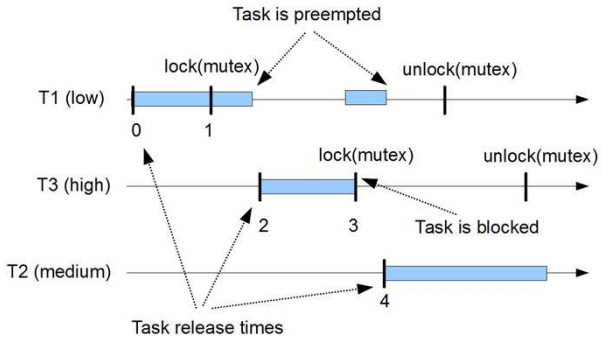
Fixed priority and shared resources

- Shared resources are modeled by semaphores for scheduling analysis.
- **We use specific semaphores implementing inheritance protocols:**
  - To take care of priority inversion.
  - To compute worst case task waiting time for the access to a shared resource. Blocking time Bi.

- **Inheritance protocols:**
  - PIP (Priority inheritance protocol**)**, can not be used with more than one shared resource due to deadlock.
  - PCP (Priority Ceiling Protocol) , implemented in most of real-time operating systems (e.g. VxWorks).
  - Several implementations of PCP exists: OPCP, ICPP, …

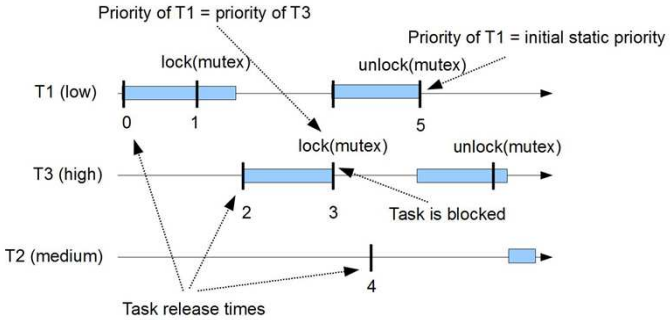
Fixed priority and shared resources

- **What is priority inversion:** a low priority task blocks a high priority task



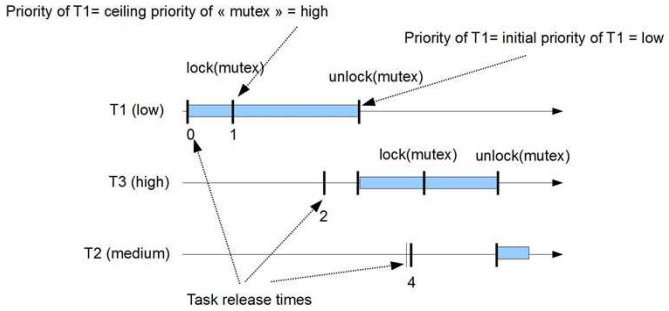
- B_i = worst case on the shared resource blocking time.

Fixed priority and shared resources



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

Fixed priority and shared resources



- **ICPP (Immediate Ceiling Priority Protocol):**
  - Ceiling priority of a resource = maximum fixed priority of the tasks which use it.
  - Dynamic task priority = maximum of its own fixed priority and the ceiling priorities of any resources it has locked.
  - B_i=longest critical section ; prevent deadlocks

Fixed priority and shared resources

- **How to take into account Bi (blocking time):**
  - Processor utilization factor test :

$$\forall i, 1 \leq i \leq n : \sum_{k=1}^{i-1} \frac{Ck}{Pk} + \frac{Ci+Bi}{Pi} \leq \; i.(2^{\frac{1}{i}} - 1)$$

  - Worst case response time :

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

To conclude on scheduling analysis

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

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

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

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

$$R_i = w_i + J_i$$

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

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

Many feasibility tests …. Many assumptions …

How to choose them?

Summary

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

AADL to the rescue ?

- **Issues:**
  - Ensure all required model elements are given for the analysis
  - Ensure model elements are compliant with analysis requirements/assumptions
- **AADL helps for the first issue:**
  - AADL as a pivot language between tools. International standard.
  - Close to the real-time scheduling theory: real-time scheduling analysis concepts can be found. Ex:
    - Component categories: thread, data, processor
    - Property: `Deadline, Fixed Priority, ICPP, Ceiling Priority, …`

Property sets for scheduling analysis

- **Properties related to processor 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, ..
```

Property sets for scheduling analysis

- **Properties related to the threads/data components:**

```
Compute_Execution_Time: Time_Range
 applies to (thread, subprogram, …);

Deadline: inherit Time => Period applies to (thread, …)

Period: inherit Time applies to (thread, …);

Dispatch_Protocol: Supported_Dispatch_Protocols
 applies to (thread);
--  Periodic, Sporadic, Timed, Hybrid, Aperiodic, Backg
...

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

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


AADL to the rescue ?

- **Issues:**
  - Ensure all required model elements are given for the analysis
  - Ensure model elements are compliant with analysis requirements/assumptions
- **And for the second issue?**

Cheddar : a framework to access schedulability of AADL models

- **Cheddar tool =** analysis framework (queueing system theory & real-time scheduling theory)
  + internal ADL (analysis model)
  + simple analysis model editor
  + optimization tools (partitioning)
  + various ADL inputs (AADL, MARTE UML, …)
  + …
- **Two versions :**
  - Open source (Cheddar) : educational and research.
  - Commercial product (AADLInspector) : Ellidiss Tech product.
- **Supports :** Ellidiss Tech., Conseil régional de Bretagne, Brest Métropole, EGIDE/Campus France, Thales Communication, BPI France

Cheddar : main analysis/modeling features

- **Analysis by scheduling simulations:**
  - With preemptive and non preemptive scheduling policies.
  - With uniprocessor and multiprocessor policies: Rate Monotonic, Deadline Monotonic, Least Laxity First, Earliest Deadline First, POSIX queueing policies, Maximum Urgency First, Round-Robin, time sharing scheduling policies. Proportionate Fair, EDZL, LLREF, hierarchical schedulers such as ARINC 653 scheduling, sporadic server, polling server or deferable server.
  - With instruction cache entities and the several CRPD computation models.
  - Aperiodic task, periodic task, sporadic task, task activated according to a poisson process or a user-defined policy.
  - Shared resources : FIFO, PCP, PIP, IPCP.
  - With task jitters and offsets.
  - With various task dependency constraints.
- **Sustainable scheduling simulation with cache unit with a dedicated CRPD model.**
- **Scheduling analysis with user-defined policies.**
- **Extract data from scheduling simulations, such as:**
  - Worst/best/average task response times, task missed deadlines.
  - Number of preemption, number of context switches.
  - Worst/best/average shared resource blocking time.
  - Deadlock and priority inversion.
  - Worst/average buffer utilization factor, message worst/average waiting time.

Cheddar : main analysis/modeling features

- **Task schedulability/feasibility tests :**
  - Compute worst case task response times on periodic tasks set by methods similar to Joseph and Pandia (for any deadline/period, for preemptive and non preemptive scheduling policies, for dynamic or static scheduling policies, with jitter).
  - Compute worst case response times with linear and tree transactions : Tindell, Audsley, WCDOPS+Plus and WCDOPS+_NIMP methods.
  - Apply processor utilization feasibility tests.
  - Memory footprint of software entities.
- **Worst case shared resources blocking time analysis:** FIFO, PIP, OPCP, IPCP.
- **CRPD computation:** UCB, ECB, UCB-Union, UCB-union-Multiset, ECB-Union-Multiset, combined Multiset.
- **Queueing system theory models for buffer feasibility tests:** M/M/1, M/D/1, P/P/1 M/P/1
- **Tools to express and perform analysis with task dependencies:**
  - Model task transactions (linear or tree) and compute worst case response times.
  - Compute Scheduling simulations according to task precedencies.
  - Compute Tindell Holistic end to end response time.
  - Apply Chetto and Blazewicz algorithms on task deadlines.

Cheddar : main analysis/modeling features

- **Design space exploration with PAES**: with also an example to cluster tasks
- **Task and resource priority assignments:**
  - Classical Rate Monotonic, Deadline Monotonic, Audsley task priority assignments.
  - Task priority assignment according to CRPD.
  - Shared resource ceiling priority assignment (for PCP policies).
- **Partitioning algorithms for periodic task set:** Best fit policy, General Task fit policy, Fist fit policy, Small fit policy, Next fit policy
- Features to automatically generate Cheddar anaysis models according to UUNIFAST or similar algorithms.

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:**

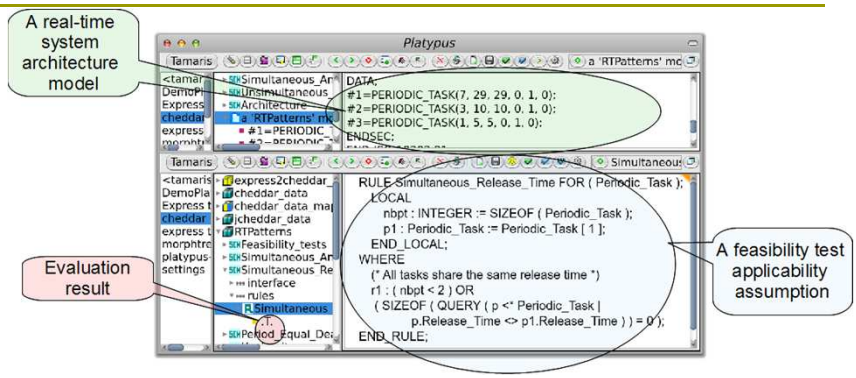
    = models a typical thread communication or synchronization + a typical execution platform

    = 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.

Design pattern compliancy verification



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

Example : «Ravenscar» design pattern

- **Specification of various design patterns:**
  - **Time-triggered :** sampling data port communication between threads
  - **Ravenscar** : PCP shared data communication between threads
  - **Queued buffer/ARINC653 :** producer/consumer synchronization
  - **Black board/ARINC653 :** readers/writers synchronization
  - **…**
  - **Compositions of design patterns.**
- **Ravenscar:** used by TASTE/ESA
- **Constraints defining "Ravenscar" to perform the analysis with a given schedulability test:**
  - Constraint 1 : all threads are periodic
  - Constraint 2 : threads start at the same time
  - Constraint 3 : shared data with PCP
  - Constraint n : fixed preemptive priority scheduling + uniprocessor
  - …

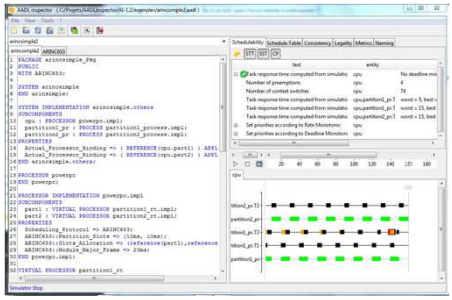
Example : «Ravenscar» compliant AADL model

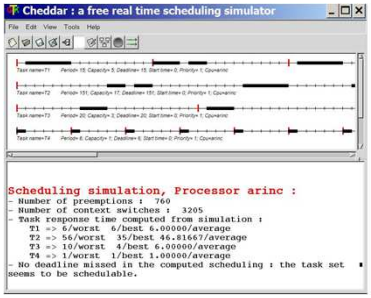
```
thread implementation receiver.impl
  properties
    Dispatch_Protocol => Periodic;
    Compute_Execution_Time => 31 ms .. 50 ms;
    Deadline => 250 ms;
    Period => 250 ms;
end receiver.impl;

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

```
process implementation processing.others
  subcomponents
    receiver : thread receiver.impl;
    analyzer : thread analyzer.impl;
    target : data target_position.impl;
    . . .
processor implementation leon2
  properties
    Scheduling_Protocol =>
          RATE_MONOTONIC_PROTOCOL;
       Preemptive_Scheduler => true;
end leon2;

system implementation radar.simple
subcomponents
  main : process processing.others;
  cpu : processor leon2;
  . . .
```


Demos, practical labs

- Scheduling analysis of the radar example with AADLInspector & Cheddar





Conclusion

To summarize

- **We introduced the concepts of AADL**
  - Architectural description
  - Patterns for scheduling analysis
- **Not covered today:**
  - Code generation => Ocarina, J. Hugues/ISAE
  - Reliability analysis using Error Modeling Annex => P. Feiler CMU/SEI
  - Modeling of IMA systems => L. Pautet/Télécom Paris, E. Borde/Télécom Paris
  - Network models & analysis => A. Khoroshilov/ISPRAS
  - Multiprocessor support & scheduling analysis => S. Rubini/Lab-STICC, F. Singhoff/Lab-STICC
  - Formal methods => B. Larson/KS Univ., J.P. Talpin/INRIA, M. Filali/IRIT
  - and much more !