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
3. AADL: tool support

Introduction

- **ADL, Architecture Description Language:**
  - **Goal :** modeling software and hardware architectures to master complexity … to perform analysis
  - **Concepts :** components, connections, configuration.
  - **Many ADLs :** formal/non formal, application domain, …
- **ADL for real-time systems:** AADL (*Architecture Analysis and Design Language*).

AADL: Architecture Analysis & Design Language

- International standard promoted by the SAE (Society of Automotive Engineers), AS-2C committee, released as AS-5506A.
- Version 1.0 published in 2004, **version 2** in 2009
- http://aadl.info list all resources around AADL
  - Public wiki with lot of resources: https://wiki.sei.cmu.edu/aadl/index.php/Main_Page
  - Include link to most research activities around AADL

- **Different representations :**
  - Graphical (high-level view of the system),
  - **Textual (to view all details),**
  - XML (to ease processing by 3rd party tool)

A 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 a semantics (natural language or formal methods).

AADL components

- **AADL model** : hierarchy/tree of components
- **AADL component:**
  - Model a software or a hardware entity
  - Has a type/interface, one or several implementations
  - May be organized in packages (reusable)
  - May have subcomponents
  - May combine/extend/refine others
  - May have properties : valued attributes (source code file name, priority, WCET, memory consumption, …)
- **Component interactions :**
  - Modeled by component connections
  - AADL features are connection points

AADL components

- **How to declare a component:**
  - Component type: name, category, properties, features.
  - Component implementation: internal structure (subcomponents), properties

- **Component categories:** model real-time abstractions, close to the implementation space (ex : processor, task, …). Each category has a well-defined semantics/behavior, refined through the property mechanism
  - Hardware components: execution platform
  - Software components
  - Systems : bounding box of a system. Model a deployment.

Component type

- AADLv2 distinguished type and implementation
- Component type = high-level specification of a component
- All component type declarations follow the same pattern:

```
<category> foo [extends <ba
features
   -- list of featu
   -- interface
properties
   -- list of prope
   --  e.g. priorit
end foo;
```

Inherit features and properties from parent

Interface of the component: (event) ports, access to data or subprograms

Some properties describing non-functional aspect of the component

Component type

- **Example:**

```
  subprogram Spg                          --  Spg is modelling a
C function, stored
  features                                --  in file "foo.c",
that takes one
    in_param : in parameter                   eter as input
  properties
    Source_Language => C;
    Source_Text => ("foo.c");
  end Spg;

  thread bar_thread                        --  bar_thread is a
sporadic thread,
  features
whenever it
    in_data : in event data po
on its"in_data"
  properties
    Dispatch_Protocol => Sporadic;
  end bar_thread;
```

Standard properties, one can define its own properties

Note: standard defines validity of combination of properties. To be complete, a sporadic thread must define a minimal Inter-arrival time

Component implementation

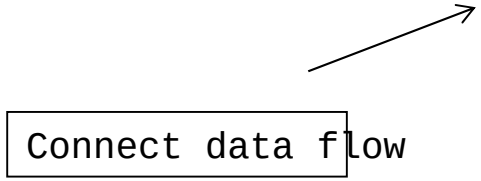
- AADLv2 distinguishes type from implementation
- Component Implementation complete the interface
  - Similar to spec/body of Ada, interface/implementation in Java

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

foo.i implements fo

Component implementation

- **Example:**




```
thread bar_thread                                -- bar_thread i
features                                         -- it is dispat
  in_data : in event data port foo_data;         -- receives an
properties                                       -- port
  Dispatch_Protocol => Sporadic;
end bar_thread;

thread implementation bar_thread.impl            -- in this impl
calls                                            -- dispatch we
  C : { S : subprogram spg; };                   -- sequence. We
connections                                      -- parameter to
  parameter in_data -> S.in_param;
end bar_thread.impl;
```

AADL concepts

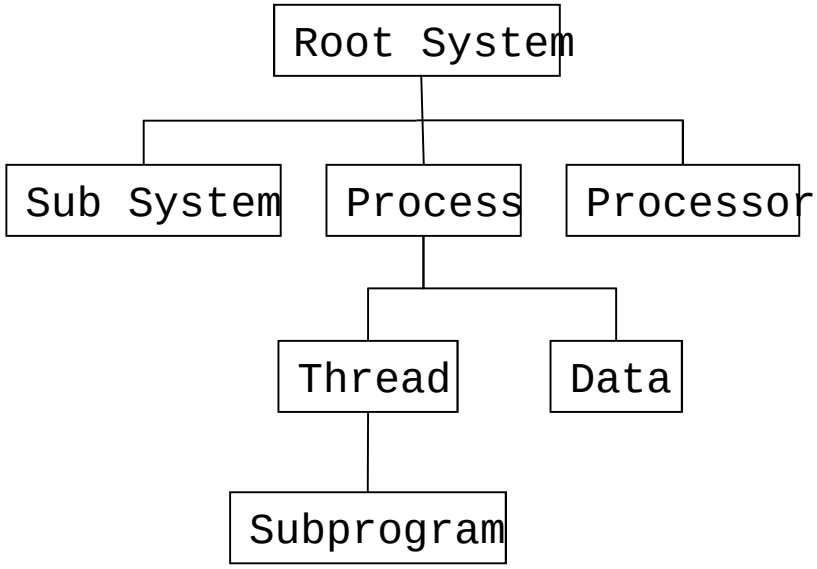
- **AADL introduces many other concepts:**
  - Related to embedded real-time distributed systems :
    - AADL flows: capture high-level data+execution flows
    - AADL modes: models an operational mode in the form of an alternative set of active components/connections/…
  - To ease models design/management:
    - AADL packages (similar to Ada, renames, private/public)
    - AADL abstract component, component extension
    - …
- **AADL is a rich language :**
  - 100 entities in the meta-model
  - BNF has 185 syntax rules
  - Around 250 legality rules and more than 500 semantics rules
  - 400 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
3. AADL: tool support

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

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

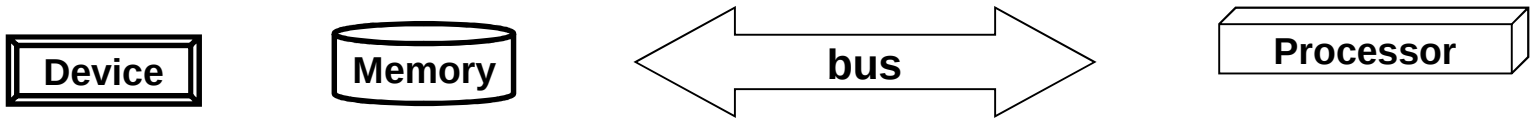
- **Example of a thread component :** a thread may call different subprograms

```
subprogram Receiver_Spg
end Receiver_Spg;

subprogram ComputeCRC_Spg
end Compute_CRCSpg;

. . .
```

```
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 :** schedule component **(**combined CPU and RTOS scheduler). A processor may contain multiple virtual processors.
- **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. Model the deployment of components inside the component hierarchy. Concept of binding.

« system » category

```
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;
```

```
device antenna
end antenna;

processor leon2
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: some restrictions apply on subcomponents
  - A hardware cannot contain software, etc

data	data, subprogram
thread	data, subprogram
thread group	data, thread, thread group, subprogram
process	thread, thread group, data
processor	Memory, virtual processor, bus, virtual processor
memory	Memory, bus
system	All except subprogram, thread et thread group

Outline

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

AADL properties

- **Property:**
  - Typed attribute, associated to one or more components
  - Property = name + type + associated components
  - Property association = property name + value.

- **Allowed types in properties:**
  - `aadlboolean`, `aadlinteger`, `aadlreal`, `aadlstring`, `enumeration, classifier` (component, connection, etc.), `reference` (component…), `list of …`

- Can be propagated to subcomponents: **inherit**
- Can override parent's one, case of extends

AADL properties

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

- **Example :**

```
property set Thread_Properties is
  . . .
  Deadline : 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

```
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 are associated to a component type (1) or implementation (2), as part of a subcomponent instance (3), or a contained property association (4).

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

thread implementation receiver.impl
properties – (2)
   Deadline => 150 ms;
   Period => 160 ms,
end receiver.impl;
```

```
process implementation processing.others
subcomponents
  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
3. AADL: tool support

Component connection

- **Component connection:** model component interactions, control flow and/or data flow. E.g. exchange of message, remote call (RPC),
- ***features* :** component 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, interruption)
  - *data port/event data port* : synchronous/asynchronous exchange of data/message
  - *subprogram parameter*
  - *data access* : access to a data, 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

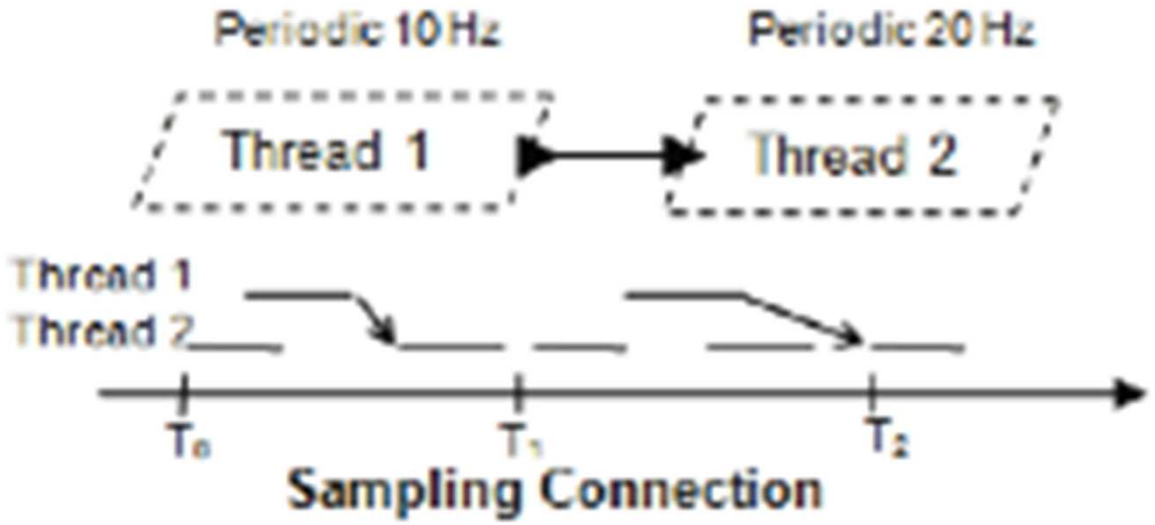
```
thread analyser
features
  analyser_out : out data port
       Target_Position.Impl;
end analyser;

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

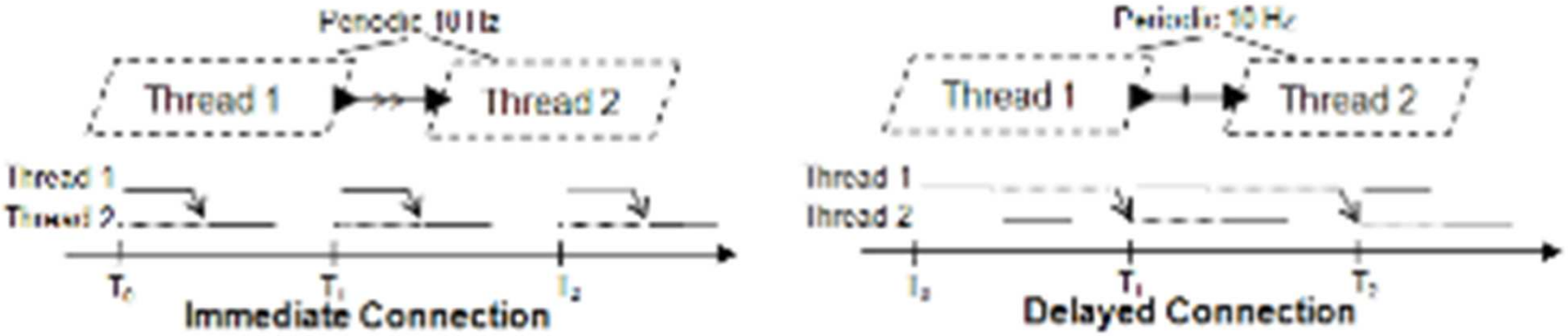
- **Multiple policies exist to control production and consumption of data by threads:**

1. **Sampling connection:** takes the latest value
   - Problem: stability of control/command algorithm



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
    1. These two policies allow for deterministic communication, hence stability of the computation



Component connection

- Thread & subprogram connection

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

```
subprogram Receiver_Spg
features
   spg_out : out parameter
      Target_Distance;
   spg_in : in parameter
      Target_Distance;
end Receiver_Spg;

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

Component connection

- **Connecting threads and shared data:**

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

Component connection

- **Thread & thread : RPC, rendez vous**

```
thread Remote
features
   MyCalc: provides subprogram access Calc;
end Remote;

thread caller
features
  MyCalc: requires subprogram access Calc;
end caller;
```

Outline

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

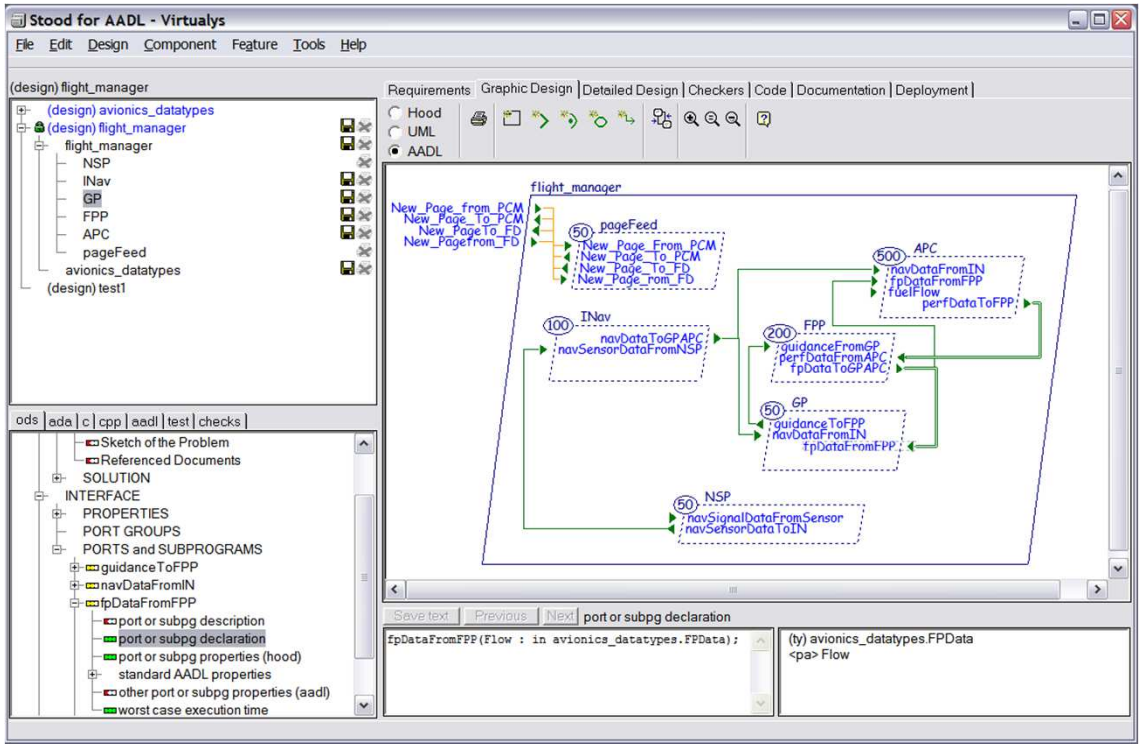
AADL & Tools

- Tools are mandatory to exploit any models
  - Otherwise, resort to traditional engineering, no value added

- **OSATE** (SEI/CMU), http://aadl.info
  - Eclipse-based tools
  - Supports the textual syntax, reference implementation
  - Supports syntactic and semantic checks
  - Some plug-ins integrated (ARINC653 patterns)
  - Support for reliability analysis (Error Modeling annex)
  - OSATE2 meta-model as a UML2 meta-model, to ease writing your own analysis or transformation plug-ins

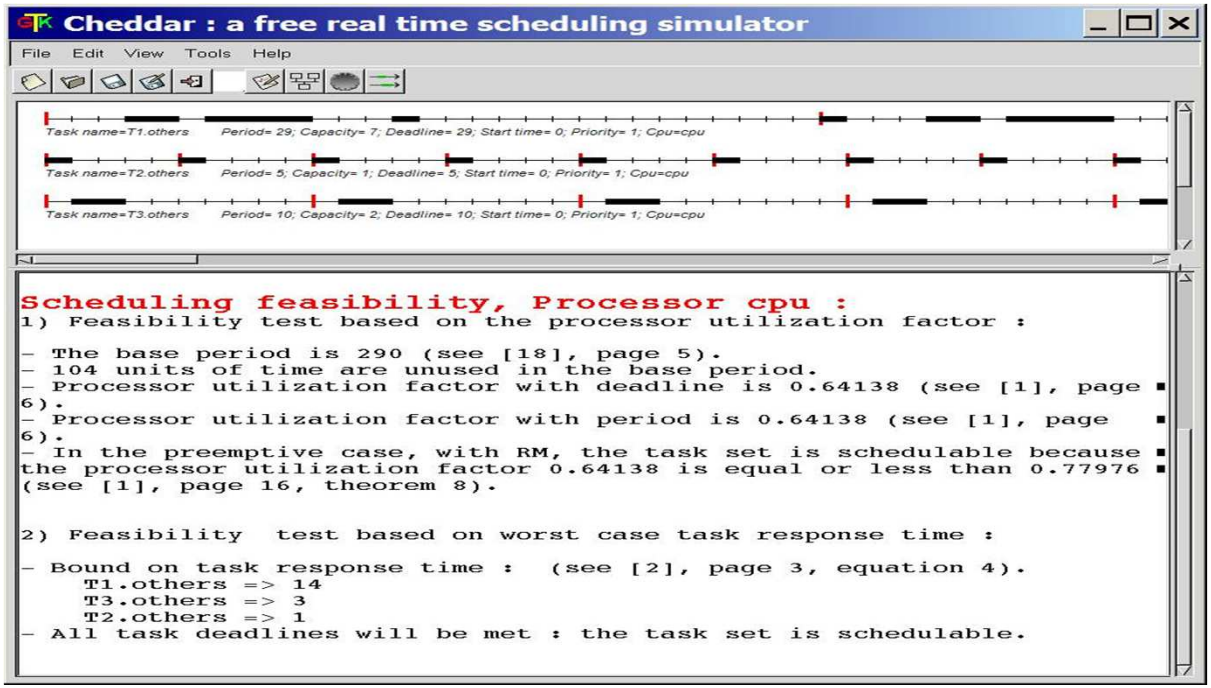
AADL & Tools

- **STOOD**, **ADELE** (Ellidiss) http://www.ellidiss.com
  - Graphical editors for AADLv1 et v2, code/documentation generation



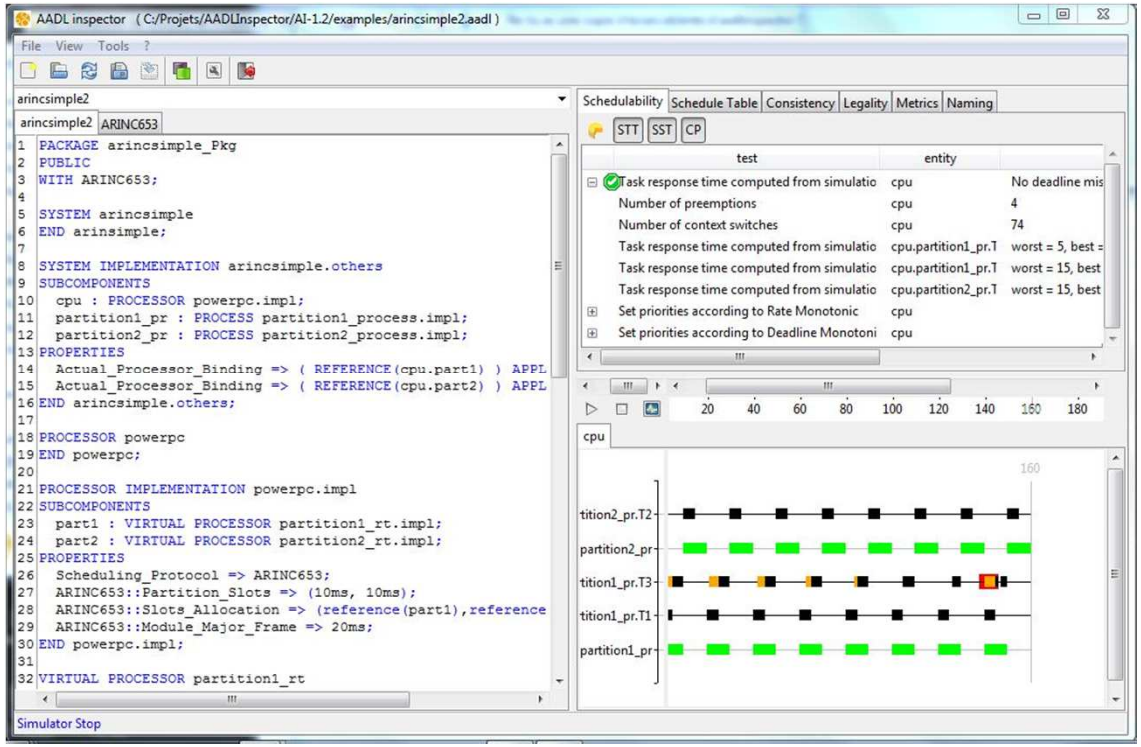
AADL & Tools

- **Cheddar** (UBO/Lab-STICC) http://beru.univ-brest.fr/~singhoff/cheddar/
  - Import of AADLV1 models
  - Performance analysis, dimensioning
  - Based on real-time scheduling theory, and queueing theory



AADL & Tools

- **AADLInspector** (Ellidiss) http://www.ellidiss.com
  - Lightweight tool to inspect AADL models, in text form
  - Connection with Cheddar, Simulation Engine,
  - AADLv2 only



AADL & Tools

- **Ocarina** (ISAE -- http://www.openaadl.org)
  - Command line tool, library to manipulate AADL models
  - AADLv1 & v2 parser, analyzer
  - Code generation for High-Integrity system, in C and Ada
    - Support for native, RTOS (RTEMS, RT-Linux), bare boards
  - Mapping to colored or timed Petri Nets, WCET, …