AADLv2 Cheat Sheet: Basics

1 AADL Components

1.1 Taxonomy of model elements

`component category`	class: e.g. thread, data, ...
`component type`	features, flows, properties.
`component implementation`	subcomponents, connections call sequence, flow impl. and properties.
`packages`	Namespace for components.
`property sets`	Non-functional properties.

1.2 Component categories

`data`	data type
`thread`	execution flow
`thread group`	group of threads
`subprogram`	sequential flow
`process`	address space
`memory`	storage area
`bus`	physical interconnect
`processor`	execution resource
`device`	I/O
`virtual processor`	partition
`virtual bus`	logical interconnect
`system`	encapsulation block
`abstract`	component template

1.3 Component type declaration

```
<component_category> <component_identifier>
     extends <component_type_identifier'>
features
   -- some features
properties
   -- some properties
end <component_identifier>;
```

`extends`	inherit from another component type.
`features`	list ports, access to other elements.
`properties`	NFP properties for this type.

1.4 Component implementation declaration

```
<component_category> implementation
 <component_identifier>
   extends <component_implementation_identifier'>
subcomponents
   --
calls
   -- some call sequences
connections
   --
properties
   -- some properties
flows
modes
end <component_identifier>;
```

`subcomponents`	subcomponents (internals)
`extends`	inherit from another component type.
`calls`	call sequence (only thread and subprograms)
`connections`	link subcomponents
`properties`	NFP properties for this type.
`flows`	flow implementation
`modes`	mode automata

1.5 Packages

```
package <package_identifier>
public
   with <package_identifier_1>;
   renames <package_idenfier_1>::all;
   --  imports all entities
   renames <package_idenfier_1>::Id;
   --  partial import
   Id2 renames <package_idenfier_1>::Id2;
   --  partial renaming
end <package_identifier>;
```

`with`	import entities
`public`	all entities are visible, need qualification
`private`	hide implementation
`renames`	full import, no qualification

1.6 Component Features

`data port`	unqueued information
`event (data) port`	pure or type signal, queued
`port direction`	in (reception), out (emission), in out
`requires access`	need to access a data or spg
`provides access`	

1.7 Features connections

Connections are directional:

```
connections
  parameter <parameter_out> -> <parameter_in>
  port <port_out> -> <port_in>
  access <provider> -> <requirer>
```

2 AADL Properties

2.1 Property sets

```
property set <ps_identifier>
  -- ...
end <ps_identifier>;
```

2.2 Pre-defined property sets:

`deployment_properties`	binding constraints
`thread_properties`	dispatching, concurrency, ...
`timing_properties`	support of time
`communication_properties`	communication mechanisms
`memory_properties`	use of storage areas
`programming_properties`	link to implementation
`modeling_properties`	name resolution rules
`aadl_project`	project-specific constants

2.3 deployment_properties

```
property set Deployment_Properties is
  Actual_Processor_Binding:
   inherit list of reference
    (processor, virtual processor)
  applies to (thread, thread group, process,
   system, virtual processor, device);
  --  Binding components to a processor

  Actual_Memory_Binding:
        inherit list of reference (memory)
  applies to (thread, thread group,
   process, system, processor, device, data,
   data port, event data port, feature group,
   subprogram);
  --  Binding components to a memory

  Actual_Connection_Binding:
   inherit list of reference (processor,
    virtual processor, bus,
    virtual bus, device, memory)
   applies to (port, connection, thread group,
    process, system, virtual bus);
  --  Binding components to a connection

  Scheduling_Protocol: inherit list of
   Supported_Scheduling_Protocols
    applies to (virtual processor, processor);
  --  See aadl_project for a full list
end Deployment_Properties;
```

2.4 thread_properties

```
property set Thread_Properties is
  Dispatch_Protocol: Supported_Dispatch_Protocols
   applies to (thread);
  -- Periodic, Sporadic, Aperiodic, Timed,
  -- Hybrid, Background

  Priority: inherit aadlinteger
   applies to (thread, thread group,
    process, system, device, data);

  Concurrency_Control_Protocol:
   Supported_Concurrency_Control_Protocols
    applies to (data);
end Thread_Properties;
```

2.5 timing_properties

```
property set Timing_Properties is
  Time: type aadlinteger 0 ps .. Max_Time
   units Time_Units;
  Time_Range: type range of Time;

  Compute_Deadline: Time
   applies to (thread, device, subprogram,
    event port, event data port);
  --  Deadline of an activity

  Compute_Execution_Time: Time_Range
   applies to (thread, device, subprogram,
    event port, event data port);
  --  Worst Case Execution Time
  Period: inherit Time
   applies to (thread, thread group,
    process, system, device);

 Timing: enumeration
  (sampled, immediate, delayed)
   => sampled applies to (port);
 --  Timing of communications
end Timing_Properties;
```

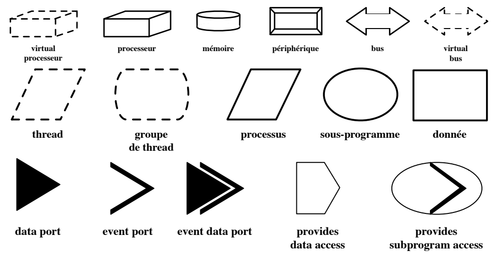
2.6 communication_properties

```
property set Communication_Properties is
  Overflow_Handling_Protocol: enumeration
    (DropOldest, DropNewest, Error)
   => DropOldest
  applies to (event port,
   event data port, subprogram);

  Queue_Processing_Protocol:
   Supported_Queue_Processing_Protocol => FIFO
   applies to (event port,
    event data port, subprogram);

  Queue_Size: aadlinteger 0 .. Max_Queue_Size => 1
   applies to (event port,
    event data port, subprogram);
end Communication_Properties;
```

2.7 programming_properties

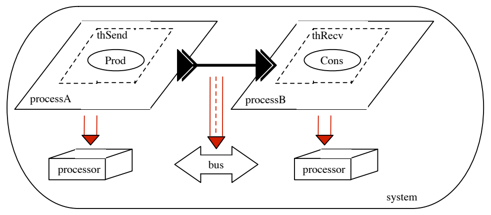
```
property set Programming_Properties is
  Compute_Entrypoint:
   classifier (subprogram classifier)
  applies to (thread,
   subprogram, event port, event data port);
 --  Also Initialize_, Recovery_, ..

 Source_Language: inherit
  Supported_Source_Languages
   applies to (subprogram, data,
    thread, thread group, process, system,
    bus, virtual bus, device, processor);

 Source_Text: inherit list of aadlstring
  applies to (data, port,
   virtual bus, subprogram, thread,
   thread group, process, system,
   memory, bus, device, processor,
   parameter, feature group, package);
end Programming_Properties;
```

3 Graphical AADL notations



4 Example: Producer/Consumer

This models proposes a simple producer/consumer pattern, covering most AADLv2 constructs from this sheet.



```
package Software
public
 with Data_Model;

 data Simple_Type
 properties
  Data_Model::Data_Representation => integer;
 end Simple_Type;

 subprogram Do_Ping_Spg
 features
  Data_Source : out parameter Simple_Type;
 end Do_Ping_Spg;

 subprogram Ping_Spg
 features
  Data_Sink : in parameter Simple_Type;
 end Ping_Spg;

 thread thSend
 features
  Data_Source : out event data port Simple_Type;
 end thSend;

 thread implementation thSend.Impl
 calls
  Mycall : { P_Spg : subprogram Do_Ping_Spg; };
 connections
  parameter P_Spg.Data_Source -> Data_Source;
 properties
  Dispatch_Protocol => Periodic;
  Period            => 1000 ms;
  Priority          => 2;
 end thSend.Impl;

 thread thRecv
 features
  Data_Sink : in event data port Simple_Type;
 end thRecv;

 thread implementation thRecv.Impl
 connections
   parameter Data_Sink -> Q_Spg.Data_Sink;
 properties
   Dispatch_Protocol  => Sporadic;
   Period             => 10 ms; -- MIAT
   Priority           => 1;
   Compute_Entrypoint => classifier (Ping_Spg);
 end thRecv.Impl;
end Software;
```

```
package PING_Package
public
 with software;
 with deployment;

 processor the_processor
 features
  ETH : requires bus access Ethernet_Bus;
properties
  Scheduling_Protocol =>
   POSIX_1003_HIGHEST_PRIORITY_FIRST_PROTOCOL;
 end the_processor;

 bus Ethernet_Bus end Ethernet_Bus;

 process A
 features
  Out_Port : out event data port Software::Simple_Type;
 end A;

 process implementation A.Impl
 subcomponents
   Pinger        : thread Software::thSend.Impl;
 connections
   port Pinger.Data_Source -> Out_Port;
 end A.Impl;

 process B
 features
  In_Port  : in event data port Software::Simple_Type;
 end B;

 process implementation B.Impl
 subcomponents
   Ping_Me        : thread Software::thRecv.Impl;
 connections
  port In_Port -> Ping_Me.Data_Sink;
 end B.Impl;

 system PING end PING;

 system implementation PING.Native
 subcomponents
  Node_A : process A.Impl;
  Node_B : process B.Impl;

  CPU : processor the_processor;
  the_bus : bus Ethernet_Bus;

 connections
  bus access the_bus -> CPU.ETH;
  port Node_A.Out_Port -> Node_B.In_Port
   {Actual_Connection_Binding => (reference (the_bus));};
 properties
  Actual_Processor_Binding =>
     (reference (CPU)) applies to Node_A;
  Actual_Processor_Binding =>
     (reference (CPU)) applies to Node_B;
 end PING.Native;

end PING_Package;
```