What is an AADL Subset ?

V. Gaudel†, P. Dissaux*, A. Plantec†,
F. Singhoff†, J. Hugues**, J. Legrand*

†University of Brest/UBO, Lab-Sticc, France
*Ellidiss Technologies, France
** Institut Supérieur de l'Aéronautique et de l'Espace/ISAE, France

February, 2013

Introduction (1/2)

Rationale for the Subset annex (February 2012 Meeting)

1. AADL is a rich Language.
2. Each verification/code generation may have specific requirements.
3. Tools that are devoted for a given analysis usually support a subset of AADL.

Addressed problems

1. Use of AADL may lead to some tool interoperability failures.
2. Probably causes a limited use of some AADL tools.

Objectives of the Subset annex (February 2012 Meeting)

1. Increase tool interoperability.
2. Increase confidence of users when they (try to) use tools.
3. Certification toolkits for subset: allow tool designers to check compliance with their products.
4. Allow users to define constraints that are specific to their systems or overall development process.

Introduction (2/2)

Problems we try to answer (February 2012 Meeting)

1. What is a subset?
2. How to express it?

Proposition

1. Investigate 3 examples of Subsets.
2. Proposition of a superset from whom all subsets could be defined.
3. Investigate the different kinds of constraints of those subsets.
4. Proposition of an uniform way to describe constraints.

Outline

1. Subset Examples
2. Superset: an AADL Meta-Model
3. Examples of cardinality constraints
4. Mapping towards REAL and Prolog
5. Conclusion

Subset Example 1: Marzhin V1

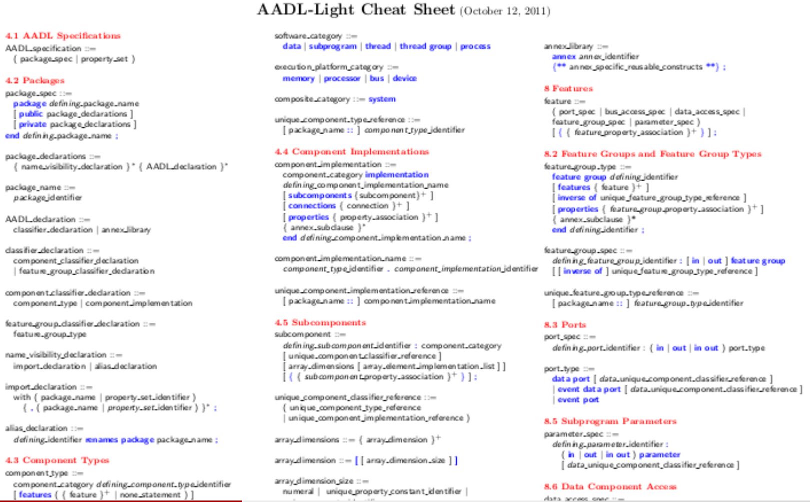
- **Require**: There is only one Processor component.
- **Require**: The property Actual_Processor_Binding must be specified.
- **Require**: For all processors, property Scheduling_Protocol must have the following values: *POSIX_Fixed_Priority_Scheduling_Protocol*, *Rate_Monotonic_Protocol* or *Deadline_Monotonic_Protocol*.
- **Require**: The property Dispatch_Protocol must have one of the following values: Periodic, Aperiodic, [...], Background.
- **Require**: Properties must be one of the following: *Dispatch_Protocol*, Period, Deadline, Priority, *Compute_Execution_Time*
- ...

Subset Example 2: AADL-Light (BLESS Update of October 2012).

- **Authorized**: See AADL-Light Cheat Sheet.

- **Forbid**: **There is no** abstract component.
- **Forbid**: **There is no** subprogram call sequence.
- **Forbid**: **There is no** in-binding.
- **Forbid**: **There is no** contained property association.
- ...

Subset Example 2: AADL-Light (BLESS Update of October 2012).



AADL-Light Cheat Sheet (October 12, 2011)

4.1 AADL Specifications

```
AADL_specification ::=
  ( package_spec | property_set )
```

4.2 Packages

```
package_spec ::=
  package defining_package_name
  [ public package_declarations ]
  [ private package_declarations ]
end defining_package_name ;

package_declarations ::=
  { name_visibility_declaration }* { AADL_declaration }*

package_name ::=
  package_identifier

AADL_declaration ::=
  classifier_declaration | annex_library

classifier_declaration ::=
  component_classifier_declaration
  | feature_group_classifier_declaration

component_classifier_declaration ::=
  component_type | component_implementation

feature_group_classifier_declaration ::=
  feature_group_type

name_visibility_declaration ::=
  import_declaration | alias_declaration

import_declaration ::=
  with ( package_name | property_set_identifier )
    { , ( package_name | property_set_identifier ) }* ;

alias_declaration ::=
  defining_identifier renames package package_name ;
```

4.3 Component Types

```
component_type ::=
  component_category defining_component_type_identifier
  [ features ( { feature }+ | none_statement ) ]

software_category ::=
  data | subprogram | thread | thread group | process

execution_platform_category ::=
  memory | processor | bus | device

composite_category ::= system

unique_component_type_reference ::=
  [ package_name :: ] component_type_identifier
```

4.4 Component Implementations

```
component_implementation ::=
  component_category implementation
  defining_component_implementation_name
  [ subcomponents {subcomponent}+ ]
  [ connections { connection }+ ]
  [ properties { property_association }+ ]
  { annex_subclause }*
  end defining_component_implementation_name ;

component_implementation_name ::=
  component_type_identifier . component_implementation_identifier

unique_component_implementation_reference ::=
  [ package_name :: ] component_implementation_name
```

4.5 Subcomponents

```
subcomponent ::=
  defining_subcomponent_identifier : component_category
  [ unique_component_classifier_reference ]
  [ array_dimensions [ array_element_implementation_list ] ]
  [ { { subcomponent_property_association }+ } ] ;

unique_component_classifier_reference ::=
  ( unique_component_type_reference
  | unique_component_implementation_reference )

array_dimensions ::= { array_dimension }+

array_dimension ::= [ [ array_dimension_size ] ]

array_dimension_size ::=
  numeral | unique_property_constant_identifier |

annex_library ::=
  annex annex_identifier
  {** annex_specific_reusable_constructs **} ;
```

8 Features

```
feature ::=
  ( port_spec | bus_access_spec | data_access_spec |
  feature_group_spec | parameter_spec )
  [ { { feature_property_association }+ } ] ;
```

8.2 Feature Groups and Feature Group Types

```
feature_group_type ::=
  feature group defining_identifier
  [ features { feature }+ ]
  [ inverse of unique_feature_group_type_reference ]
  [ properties { feature_group_property_association }+ ]
  { annex_subclause }*
  end defining_identifier ;

feature_group_spec ::=
  defining_feature_group_identifier : [ in | out ] feature group
  [ [ inverse of ] unique_feature_group_type_reference ]

unique_feature_group_type_reference ::=
  [ package_name :: ] feature_group_type_identifier
```

8.3 Ports

```
port_spec ::=
  defining_port_identifier : ( in | out | in out ) port_type

port_type ::=
  data port [ data_unique_component_classifier_reference ]
  | event data port [ data_unique_component_classifier_reference ]
  | event port
```

8.5 Subprogram Parameters

```
parameter_spec ::=
  defining_parameter_identifier :
    ( in | out | in out ) parameter
    [ data_unique_component_classifier_reference ]
```

8.6 Data Component Access

```
data_access_spec ::=
```

Subset Example 2: AADL-Light (BLESS Update of October 2012).

- **Authorized**: See AADL-Light Cheat Sheet.

- **Forbid**: **There is no** abstract component.
- **Forbid**: **There is no** subprogram call sequence.
- **Forbid**: **There is no** in-binding.
- **Forbid**: **There is no** contained property association.
- ...

Subset Example 3: Cheddar Subsets

- **Require**: For all threads: Dispatch_Protocol must be set to Periodic.
- **Require**: All connections must be Data Port connections.
- **Forbid**: There is no data component.
- **Forbid**: All features must be Data Port.
- **Forbid**: For all Data port, property Timing must have the following values only: *sampled*, *immediate* or *delayed*.
- **Require**: If property *Concurrency_Control_Protocol* has the values *Priority_Ceiling_Protocol* or *Immediate_Priority_Ceiling_Protocol*, Data Ceiling priority must be higher or equal to the maximum value of property Priority of all threads connected to the data component.
- ...

Different ways to define subsets:

- Subset: AADL-Light
    - AADL Declarative Model
    - Specifies Authorized/Forbidden parts
- Subsets: Cheddar, Marhzin V1
    - AADL instance model
    - Specifies Restrictions parts.

But of course, they have a common point: AADL Meta-model.

Outline

1 Subset Examples

2 Superset: an AADL Meta-Model

3 Examples of cardinality constraints

4 Mapping towards REAL and Prolog

5 Conclusion

Rationale for the SuperSet Meta-Model

Superset: a meta-model common to all subsets

1. Based on Appendix C for element identifiers
2. And literal descriptions of entities' attributes
3. Use of multiple inheritance
4. **What is in the superset?**
    - Model of the declarative part of AADL.
    - Instance model can be deduced from this model.
    - Property sets and annexes are considered as parts of the superset.

Meta Model Specification with Platypus

Use of Platypus for prototyping

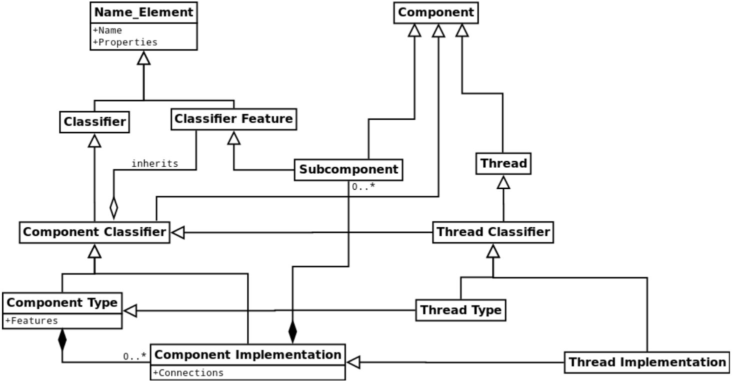
- Meta-environment based on ISO STEP technology.
- Enables to design, to verify and to validate meta-models written with EXPRESS.
- Enables to implement code generators for EXPRESS meta-model.
- Meta-model elaboration within Platypus
  - EXPRESS is readable
  - The model is checked and evaluated during design
  - Enables multiple inheritance
  - Platypus is already used for code generation with Cheddar
  - We can specify metrics
  - Definition of rules to implement consistency rules
  - Possibility of using this kind of rule for subset definition

What could be a subset?

New Subset Model Proposal

1. Superset is an EXPRESS Meta-model
2. A subset constraint is modeled by an EXPRESS RULES on the superset
3. Then, each subset is declared as a set of EXPRESS RULES on the superset
4. What we assume:
   - A constraint is a cardinality verification
   - Or a composition of cardinality verifications

Graphical Excerpt of the superset meta-model



Outline

1 Subset Examples

2 Superset: an AADL Meta-Model

3 Examples of cardinality constraints

4 Mapping towards REAL and Prolog

5 Conclusion

From literal constraints to cardinality constraints

Summary of encountered constraints:

- There is no [*model element*]
- There must be [*model element*]
- The value/content of [*model element*] must be [...]
- [*Some property*] must be specified
- For all [*model element*], [*constraint upon dependent model element*]

From literal constraints to cardinality constraints:

There must be [*model element*]

- **Forbid**: There is no data component.

```
RULE No_Data_Instance FOR ( Data_Instance );
WHERE
  R-TT-C2 : SIZEOF ( Data_Instance ) =0;
END_RULE;
```

From literal constraints to cardinality constraints

There is no [*model element*]

- **Require**: There is only one Processor component.

```
RULE Only_One_Processor FOR ( Processor_Instance );
WHERE
  RM1 : SIZEOF ( Processor_Instance ) =1;
END_RULE;
```

From literal constraints to cardinality constraints:

For all [*model element*], [*constraint upon dependent model element*]

- **Require**: For all threads, the property dispatch protocol must be periodic.

```
RULE Dispatch_Protocol_Must_Be_Periodic FOR ( Thread_Classifier );
WHERE
  RM4_Part3 : SIZEOF ( QUERY ( t <* Thread_Classifier |
              ( SIZEOF ( QUERY ( p <* t.properties |
               ( ( p.Property_Name = 'Dispatch_Protocol' ) AND
                 ( p.VALUE = 'Periodic' ) ) ) ) =0 ) ) ) =0;
END_RULE;
```

From literal constraints to cardinality constraints:

For all [*model element*], the value/content of [*model element*] must be [...]

- **Require**: For all processors, property *Scheduling_Protocol* must have the following values: *POSIX_Fixed_Priority_Scheduling_Protocol*, *Rate_Monotonic_Protocol* or *Deadline_Monotonic_Protocol*.

```
RULE Scheduling_Protocol_Must_Be_Posix_FP FOR ( Component_Classifier );
WHERE
  RM3_Part1 : SIZEOF ( QUERY ( c <* Component_Classifier |
        ( ( c.category = processor ) AND
          ( SIZEOF ( QUERY ( p <* c.properties |
                 ( ( p.Property_Name = 'Scheduling_Protocol' ) AND
                   ( p.VALUE = 'Posix_Fixed_Priority_Scheduling_Protocol' ) ) ) ) = 0 ) ) ) ) = 0;
END_RULE;

[...]
                   ( p.VALUE = 'Rate_Monotonic_Scheduling_Protocol' ) ) ) ) = 0 ) ) ) ) = 0;

[...]
                   ( p.VALUE = 'Deadline_Monotonic_Scheduling_Protocol' ) ) ) ) = 0 ) ) ) ) = 0;
END_RULE;
```

From literal constraints to cardinality constraints:

And so on ...

- **Require**: If property *Concurrency_Control_Protocol* has the value *Priority_Ceiling_Protocol*, data Ceiling priority must be higher or equal to the maximum value of property Priority of all threads connected to the data component
- **Require**: For each Data with *Concurrency_Control_Protocol* = *Priority_Ceiling_Protocol*, their Ceiling_Priority must be higher or equal to the property Priority of all threads connected to the data.

From literal constraints to cardinality constraints:

And so on ...

- **Require**: For each Data with *Concurrency_Control_Protocol* = *Priority_Ceiling_Protocol*, their *Ceiling_Priority* must be higher or equal to the property Priority of all threads connected to the data.

```
RULE Ceiling_Priority FOR ( Data );
WHERE
  RR13 : ( SIZEOF ( QUERY ( d <* Data_Classifier |
           ( SIZEOF ( QUERY ( p <* Property |
                 ( ( p.Property_Name = 'Concurrency_Control_Protocol' ) AND
                   ( p.VALUE = 'Priority_Ceiling_Protocol' ) ) ) ) =1 ) AND
           ( SIZEOF ( QUERY ( c <* Access_Connection |
                 ( ( c.accessed_component =d ) AND
                   ( SIZEOF ( QUERY ( t <* Thread_Type |
                         ( SIZEOF ( QUERY ( f <* t.features |
                               ( f =
                                 c.requiring_feature ) AND
                               ( d.ceiling_priority <
                                 t.priority ) ) ) =
                         0 ) ) ) =
   0 ) ) ) ) =  0 ) ) ) =0 );
END_RULE;
```

Outline

1 Subset Examples

2 Superset: an AADL Meta-Model

3 Examples of cardinality constraints

4 Mapping towards REAL and Prolog

5 Conclusion

Mapping towards REAL and Prolog

There is no data component

EXPRESS:

```
  RULE No_Data_Instance FOR ( Data_Instance );
  WHERE
    R_TT_C2 : SIZEOF ( Data_Instance ) = 0;
  END_RULE;
```

Prolog:

```
isSubcomponent(_,_,_,_,'DATA',_,_,_,_) -> write('error R_TT_C2'); true.
```

REAL:

```
theorem Check_R_TT_C2
  foreach s in System_Set do
   check (Cardinal (Data_Set) = 0);
end Check_R5_2;
```

- Work in progress.
- Can be produced automatically (e.g. Platypus).

Conclusion

- Problem:
  - What is a subset and how to express it?
  - Is there an uniform way to express the various examples of subsets/constraints?
- Approach:
  - Superset: an AADL meta-model to model the examples of subsets.
  - Can we express constraints of each subset as a cardinality constraint on superset?
- Results:
  - For the considered subset examples, we are able to express all their constraints as cardinality constraints on superset.
- Perspectives/roadmap:
  - Finalize translation of constraints in REAL and Prolog. Relationships with the constraint annex ⇒ next meeting?
  - Express other subsets with cardinality constraints? Oleg?
  - Cardinality may simplify ordering of subset: can we order proposed subsets?

Acknowledgement

We would like to thank Ellidiss Technologies and Region Bretagne for their support to this project.