SAE Aerospace An SAE International Group	**AEROSPACE STANDARD**	**AS5506/TBD**
		Issued 2014-02 Draft
SAE Architecture Analysis and Design Language (AADL) Annex Volume TBD: Annex TBD: Unit Relations Annex		

RATIONALE

The purpose of this annex is to provide a way to specify relations between unit types and unit type combinations. These relations can be used to automate building of conversion functions between proportional unit types.

This Architecture Analysis & Design Language (AADL) standard document was prepared by the SAE AS-2C Architecture Description Language Subcommittee, Embedded Computing Systems Committee, Aerospace Avionics Systems Division.

A.

Unit Relations Sublanguage

1. Scope

1. The Unit Relations Sublanguage Annex provides a standard AADL sublanguage extension to define relations between different Unit Types declared in the core AADL.
2. The core AADL provides a way to define a Unit Type that is a set of measurement unit identifiers with conversion factors between them.

```
Length_Unit : type units ( mm, cm => mm * 10, m => cm * 100, km => m * 1000 );
Time_Unit : type units ( sec, min => sec * 60, h => min * 60 );
```

3. There are relations between various Unit Types that cannot be expressed in the core AADL. For example, one measurement unit can represent a relation between a couple of other ones.

```
Speed_Unit : type units ( meterph, meterpsec => meterph * 3600,
                        kmeterpsec => meterpsec * 1000, kmeterph => meterph * 1000 );
```

4. Also it may happen that two different AADL models developed by two different teams define different Unit Types to represent the same physical quantity. If these models are used within another AADL model there is no way to compare properties of that types.
5. The Unit Relations Annex is aimed to provide facilities to handle the both situations.
6. Unit Relations Annex does not support relations between Units with affine dependency between them (for example, Kelvins and Fahrenheits for Temperature Units) because it is hard to define strict semantics of multiplicative operations on such Units.
7. *It would be useful to support standard systems of measurement units (for example, SI or CGS) to not redefine units from these systems manually in model for its using.*

2. Overview of Unit Relations Sublanguage Concepts

1. Basic Unit Type is a Unit Type declared in the core AADL.
2. Derived Unit Type is formed by multiplication and division of Basic Unit Types. In particular, any Basic Unit Type is actually a trivial form of a Derived Unit Type.

```
-- 'Length_Unit * Length_Unit / Time_Unit' is an example of Derived Unit Type
```

3. A normal form of a Derived Unit Type is an expression with powers, multiplications and divisions, where each Basic Unit Type is met just once.

```
-- 'Length_Unit^2 / Time_Unit' is a Derived Unit Type in a normal form
```

4. A set of measurement units of a Derived Unit Type is formed by all combinations of measurement units of the corresponding Basic Unit Types taken in a normal form.

```
-- The set of measurement units of 'Length_Unit^2 / Time_Unit' consists of
-- Length_Unit[mm]^2/Time_Unit[sec], Length_Unit[cm]^2/Time_Unit[sec],
-- Length_Unit[m]^2/Time_Unit[sec],  Length_Unit[km]^2/Time_Unit[sec],
-- Length_Unit[mm]^2/Time_Unit[min], Length_Unit[cm]^2/Time_Unit[min],
-- Length_Unit[m]^2/Time_Unit[min],  Length_Unit[km]^2/Time_Unit[min],
-- Length_Unit[mm]^2/Time_Unit[h],   Length_Unit[cm]^2/Time_Unit[h],
-- Length_Unit[m]^2/Time_Unit[h],    Length_Unit[km]^2/Time_Unit[h].
```

5. Conceptually the Unit Relations Annex sublanguage consists of two types of statements: a unit equation and a unit type independence assertion.
6. Each units equation has two meanings. The first one says that unit types of two expressions (that are the parts of equation) represent, in fact, the same physical quantity. The second one defines that those

two unitful expressions mean the same (i.e. when one of them is used somewhere, another can be used as a replacement with no sense change).

7. A unit type independence assertion helps to prevent unintended consequences of units equation definitions. The assertion enumerates unit types that are designed to be independent with each other. If an error in unit type relations makes a subset of the types dependent, the assertions checkers allow to detect it and to warn users.

8. A recommended way of using of Unit Relations Annex is to augment each unit type declaration clause that defines a derived unit type with a Unit Relations Annex subclause that explicitly describes relation between the newly defined unit type and earlier defined ones.

9. Necessity of unitful values evaluating can arise during the management of an AADL model. For example, requirements to a model can be formalized with predicates which contain some operations on property values. Property values can be expressed as unitful values. This means performing of operations on unitful values can be required.

10. Management of operations on unitful number values is more difficult than same operations on unitless numbers. For example, any two numbers can be compared for equality, summed up, subtracted and multiplied. Same operations with unitful values can be meaningless (not applicable) in some cases. These cases have to be prohibited. For example, in SI it is meaningless to sum 1 kilogram and 1 meter.

11. Meaningful operations on unitful values would operate with both a number and a unit. For example, division of “6 m” to “2 sec” would give “3 m / sec”.

12. System of unit equations defined by Unit relations annex can be modified equivalently (using algebraic properties of operations mentioned above). The process of getting equivalent systems through the series of elementary modifications is called inferring. An equation E is called *inferred from a system of equations* S1 if it is a part of some system of equations S2 that is inferred from S1.

13. The following transformations get full set of elementary modifications enough to describe each inferring in unit equations:

    13.1. Transformations basing on a single equation `D1 = D2`:

    - multiplication of both parts of an equation by the same expression (getting `D1 * D = D2 * D`, where `D` is an expression);
    - commuting parts of an equation (getting `D2 = D1`);
    - exponentiation of both parts of an equation with some power (getting `D1^n = D2^n`, where `n` is a real number);
    - replacing of an expression with power of zero with unitless value 1;
    - simplification of expressions considering distributivity and associativity of multiplication, division and exponentiation.

    13.2. Transformation basing on a couple of equations `D1 = D2` and `D3 = D4`, where `D3` is contained in `D1`. System can be enriched by an equation `E1 = D2`, where `E1` is made from `D1` by replacing of some occurrences of `D3` to `D4`.

    13.3. Transformation basing on a unit type definition. If some unit type `T` is defined to have units `m1` and `m2` with relation `m1 => m2 * C`, then any usage of unit value `T[m1]` can be replaced by `T[m2] * C` and vice versa.

3. Unit Relations Sublanguage Grammar

1. Unit Relations Annex can be declared either as an annex library or as an annex subclause attached to a property set.

2. Unit Relations Annex consists of one or more units equations and unit type independence assertions.

Syntax

```
unit_relations_annex ::=
  { unit_types_independence_assertion | unit_types_equation }+

unit_types_independence_assertion ::=
  assert independence unit_type { , unit_type }+;

unit_types_equation ::=
  unit_expr = unit_expr ;

unit_expr ::= unit_div_expr

unit_div_expr ::=
    unit_div
  | ( unit_div ) { * ( unit_div ) }+

unit_div ::=
    unit_mul [ / div_denominator ]
  | unitless_mul / div_denominator

unit_mul ::=
    { unitless_mul * }* unit_power { * ( unit_power | unitless_mul ) }*

div_denominator ::=
    unit_power
  | ( unit_mul )

unit_power ::=
  unit_value [ ^ unitless_hi ]

unit_value ::=
  unit_type [ unit ]


-- expressions with unitless value


unitless_sum ::=
  unitless_div_expr { SUM_OP unitless_div_expr }*

unitless_div_expr ::=
    unitless_div
  | ( unitless_div ) { * ( unitless_div ) }+

unitless_div ::=
  unitless_mul [ / unitless_denominator ]

unitless_denominator ::=
    unitless_power
  | ( unitless_mul )

unitless_mul ::=
  unitless_power { * unitless_power }*

unitless_power ::=
  unitless_hi [ ^ unitless_hi ]

unitless_hi ::=
    basic_unitless_value
  | ( unitless_sum )

basic_unitless_value ::=
    signed_aadlreal_or_constant
  | signed_aadlinteger_or_constant

SUM_OP ::= + | -


-- unit types

unit_type ::= unit_unique_property_type_identifier

unit ::= unit_identifier
```

1. *unit*`_unique_property_type_identifier`, *unit*`_identifier`, `signed_aadlreal_or_constant` and `signed_aadlinteger_or_constant` are syntactic categories inherited from the core AADL as is.
2. This syntax does not allow mixed use of divisions and multiplications with non-intuitive relying on left associativity. For example it is impossible to write an expression like `Power[N] = Length[m] / Time[s]^2 * Mass[kg];` and parentheses have to be used to explicitly set the priority, like `Power[N] = (Length[m] / Time[s]^2) * Mass[kg];` or `Power[N] = Length[m] / (Time[s]^2 * Mass[kg]);`. But still, this syntax does not enforce usage of parentheses when the meaning of expression is clear, e.g. `Power[N] = Mass[kg] * Length[m] / Time[s]^2;` is allowed.

Naming Rules

1. *unit*`_unique_property_type_identifier` of `unit_type` must refer to a unit type.
2. `unit` part of `unit_value` must refer to a unit identifier defined in the declaration of the unit type referred by `unit_type`.
3. unit_type must be resolved by standard scope rules from AADL (unqualified identifier means identifier from the same package as annex declared, qualification of qualified identifier must be consist with "with" clauses of the package or property set where annex declared).

Legality Rules

1. No unit type can be included in both the left and the right parts of the same equation.
2. All unit types included in the same assertion must be different.
3. Units Relation Annex may be placed to a package or a property set level only.

Consistency Rules

1. Consistency is defined for all packages and property sets of a model. A system of units equations is formed by all units relation annex subclauses and unit types declarations of all imported property sets.

2. No equations of kind `T[m] = C`, where `T` is a Base unit type, m is a unit of type `T` and `C` is a unitless value, can be inferred from the system of units equations.

3. No equations of kind `C1 = C2`, where `C1` and `C2` are unitless values which are not equal, can be inferred from the system of units equations.

4. For all equations of kind `E = C`, where `C` is a unitless value and `E` is a unitful expression, that can be inferred from the system of units equations, if `E` contains two or more units, then at least two of them must belong to unit types that are not members of some single independence assertion.

5. Any other systems of units equations are considered consistent. This means that no conflicting consequences can be derived from a consistent system.

Semantics

1. An equation means possibility of proportional conversion a value of a Derived Unit Type from the left part of the equation to a value of a Derived Unit Type from the right part of the equation, and vice versa. For example, equation Square[kmq] = Length[km] * Length[km] means if we have a value with derived unit type Length[m] * Length[m] (as a result of intermediate computations, for example) where m is a measurement unit of unit type Length such as km => m*1000; then we may compute a value of unit type Square in kmq measurement unit by multiplication (1/1000)^2 to value in unit type Length[m] * Length[m], and result value will be of unit type Square with kmq unit.
2. A system of Unit Relation Annex equations defines an equivalence relation between proportional Derived Unit Types. Equivalence of two Derived Unit Types means that there is a proportional conversion function between values of that types. The form of the conversion function is F(u) = C * u, where u - a value of one Derived Unit Type, C - unitless constant.
3. Checking of unit relations consistency is a global operation. I.e., while checking a unit relation we must concern to all the unit relations and unit type declarations we have in a model. For example, if an AADL model imports packages from different projects and that packages defines semantically equivalent unit types, Unit Relations Annex allows to define an equation between the types in the top model. In this case system of equations of the top model is a union of equations from all imported packages and equations from the model itself.

Examples

```
property set SI is

  Length_Unit : type units ( mm, cm => mm * 10, m => cm * 100, km => m * 1000 );

  Time_Unit   : type units ( sec, min => sec * 60, h => min * 60 );

  Speed_Unit  : type units ( meterph, meterpsec => meterph * 3600,
                       kmeterpsec => meterpsec * 1000, kmeterph => meterph * 1000 );
  annex unit_relations {**
      Speed_Unit[meterph] = Length_Unit[m] / Time_Unit[h];
      assert independence Length_Unit, Time_Unit;
  **}

end SI;
```

Examples

```
property set Project1_Property_Set is

  Length_Unit : type units ( mm, cm => mm * 10, m => cm * 100, km => m * 1000 );

end Project1_Property_Set;

property set Project2_Property_Set is

  Length_Unit : type units ( mm, cm => mm * 10, m => cm * 100, km => m * 1000 );

end Project2_Property_Set;

package Project3 is

  with Project1_Property_Set;
  with Project2_Property_Set;
```

```
    annex unit_relations {**
        Project1_Property_Set::Length_Unit[m] = Project2_Property_Set::Length_Unit[m];
    **}

  end Project3;
```