REPRESENTATIONS AND MEASUREMENTS IN CIS/2

By Chuck Eastman

There are several complex structures that are used repeatedly throughout CIS/2. Most of these structures come from the integrated resources that are common to all ISO-STEP product models. A limitation of using such shared definitions is that they are completely general and can be used in the representation of airplanes and cars, as well as buildings. The benefit is that they are general and can be easily integrated with models addressing piping and process plants, mechanical and electrical subsystems, and complex models as used by some architects, such as Frank Gehry, that are using CAD/CAM techniques from the aerospace industry.

All objects are defined in STEP separately from their various representations. Shape and material properties, especially are considered representations. Representations have a context that can be used to define the units used in defining geometry and material properties globally (they can also be defined locally.)

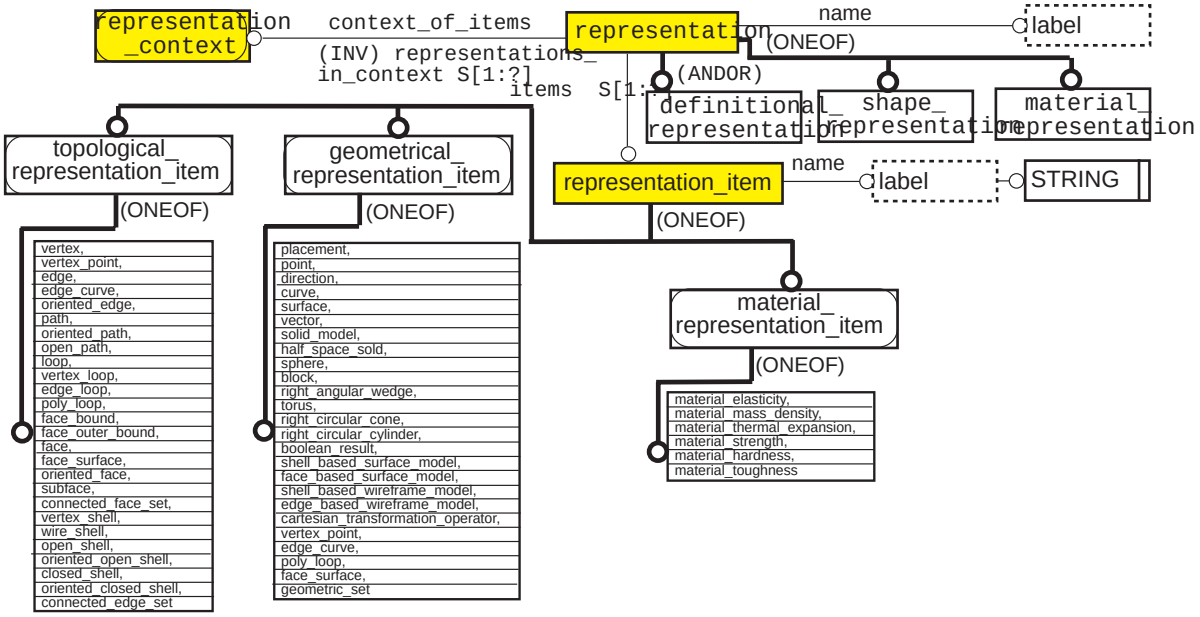

Figure One: The EXPRESS-G model of the high-level Entities defining a CIS/2 structural model.

High Level Geometry and Material Definitions

All representation entities in CIS/2 are initiatedas a representation, as shown in Figure One. Representations are classified as definitional_representaitions, shape_representations or material_representations. They all inherit the same attributes from their supertype A definitional_representation is a parametric representation.

A representation is a grouping of representation_items that all have the same context. Representation_item has three subtypes: material_ representation_item, geometrical_ representation_item and topological_ representation_item, These classify and distinguish entities that define material, simple shapes, or complex shapes, respectively. The material_

representation_items will be grouped into various material_representations, while geometrical_ and topological_representation_items will be grouped as shape_ representations or definitional_representations.

Below each type of representation_item in Figure One are large sets of more detailed entities of that general type. These are diagrammed in a compact manner; each of the Entities listed in the box below topological_representation_item are subtypes of it. The same applies to those under geometrical_representation_item and for material_representation_item. Elements of interest include point and curve.

All of these subtypes inherit a name label fromrepresentation to identify the set using the same representation_context and another to identify the representation_item. They also inherit the reference to representation_context, shown at the left of Figure One. It defines the units and measurement types that areused in the representation_items grouped by representation. Dimensions may be defined both globally and at lower levels of detail. This potentially allows an analysis model or design model to be laid out in one set of units and coordinate systems and manufacturing assemblies in another.

Measures and Units

One of the trickier aspects of ISO-STEP models is the handling of units and measures. They are defined in a standard manner for ALL step models (making every user of a STEP model pay the cost of its complexity).

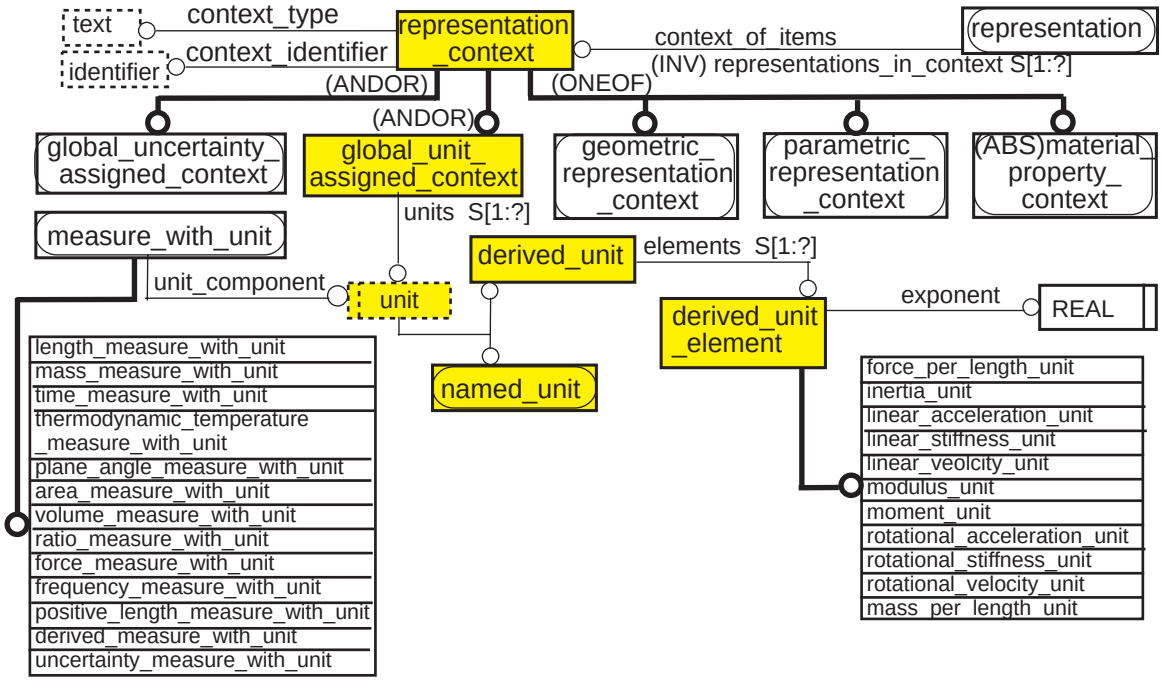

Figure Two: The EXPRESS-G model of the detail structure of representation_context.

Units are defined globally through the representation_context entity, defined at the top level of a model. Its EXPRESS-G organization is shown in Figure Two.

Representation_context has two sets of subtypes. One classifies the type of representation: geometric, parametric or material. Each provides the appropriate units for that type of

representation. Additional optional multiple subtypes (using ANDOR) are global_unit_assigned_context , which provides global default units for all measures without local units and global_uncertainty_assigned_context, which provides tolerance information for imprecise measurements. We develop the globally_assigned_context below. CIS/2 uses measure_with_unit extensively. We show that example also.

The global_unit_assigned_context has a single attribute of a set of units. A unit is a select type, meaning it may be either a named_unit or a derived_unit. Both are used in CIS/2. The main subtype of named_unit is si_unit and optionally a type of unit -- length, mass, plane angle etc. Each named_unit has a dimensions attribute giving the expected exponent. CIS/2 relies on a large number of derived units, dealing with forces, stiffness, moments and velocity. The derived_unit has a derived_unit_element (value) and is subtyped into a set of specific, derived units. They are shown is a shorthand format at the lower right of Figure Three. (Not all possible exponents are shown in Figure Two; neither are all the types of named units.)

The corresponding EXPRESS is shown below.

```
ENTITY representation_context;   -- from Part 43
  context_identifier : identifier;
  context_type       : text;
INVERSE
  representations_in_context : SET [1:?] OF representation
    FOR context_of_items;
END_ENTITY;

ENTITY global_unit_assigned_context  -- from Part 43
  SUBTYPE OF (representation_context);
  units : SET [1:?] OF unit;
END_ENTITY;

ENTITY global_uncertainty_assigned_context  -- from Part 43
  SUBTYPE OF (representation_context);
  uncertainty : SET [1:?] OF uncertainty_measure_with_unit;
END_ENTITY;

ENTITY parametric_representation_context  -- from Part 43
  SUBTYPE OF (representation_context);
END_ENTITY;

TYPE unit = SELECT -- from Part 41
  (named_unit,
   derived_unit);
END_TYPE;

ENTITY derived_unit;  -- from Part 41
  elements : SET [1:?] OF derived_unit_element;
WHERE
  WR1 : ( SIZEOF ( elements ) > 1 ) OR
   (( SIZEOF ( elements ) = 1 ) AND ( elements[1].exponent <> 1.0 ));
END_ENTITY;
```

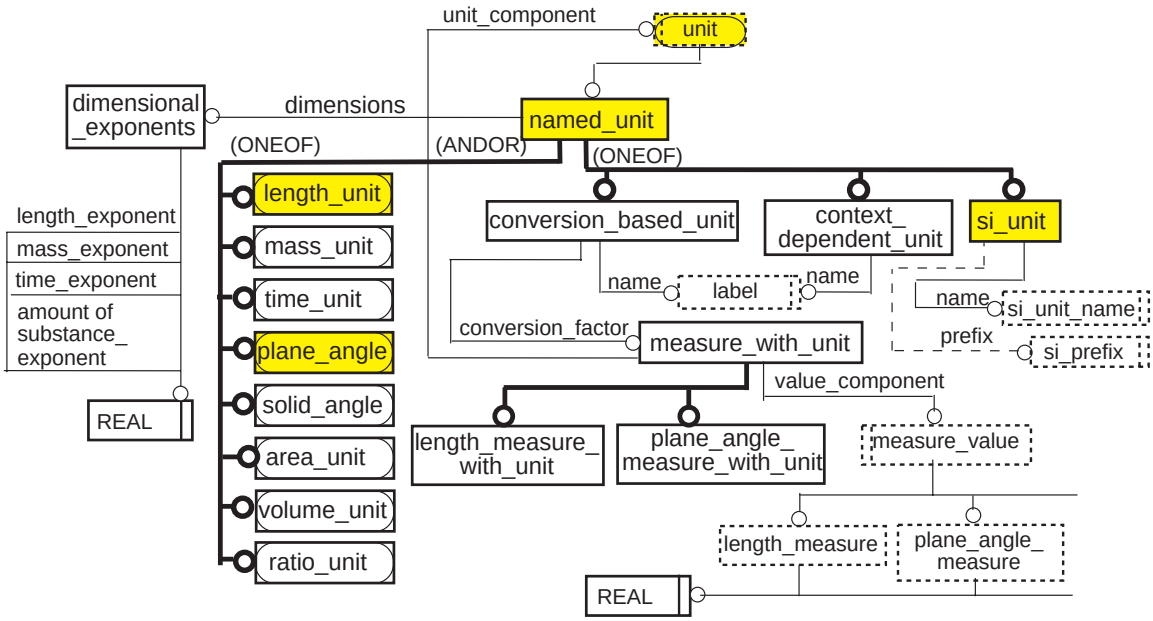

Figure Three: The more detailed structure for measures and units in Part 41.

The named_unit is defined in Figure Three. It consists of a unit and/or one of three types of units -- a conversion-based unit, a Si unit or a context dependent unit. Si units are a unit name, optionally with a prefix (.KILO., .MILLI.). A conversion_based_unit is a measure value and a named unit. These are referenced together and checked for consistency by a subtype of measure_with unit. Two such measures_with_unit subtypes are defined here, for plane angles and length.

The associated EXPRESS code is below.

```
ENTITY named_unit   -- from Part 41
   SUPERTYPE OF ((ONEOF
      (length_unit,
      mass_unit,
      time_unit,
      thermodynamic_temperature_unit,
      plane_angle_unit,
      solid_angle_unit,
      area_unit,
      volume_unit,
      ratio_unit,
      force_unit,
      pressure_unit))
         ANDOR (ONEOF (si_unit,
      conversion_based_unit,
      context_dependent_unit)) );
   dimensions : dimensional_exponents;
END_ENTITY;

ENTITY conversion_based_unit -- from Part 41
  SUBTYPE OF (named_unit);
  name              : label;
  conversion_factor : measure_with_unit;
END_ENTITY;

ENTITY context_dependent_unit   -- from Part 41
  SUBTYPE OF (named_unit);
```

```
  name : label;
END_ENTITY;

ENTITY si_unit   -- from Part 41
  SUBTYPE OF (named_unit);
  prefix     : OPTIONAL si_prefix;
  name       : si_unit_name;
DERIVE
  SELF\named_unit.dimensions : dimensional_exponents
                             := dimensions_for_si_unit (SELF.name);
END_ENTITY;


ENTITY measure_with_unit   -- from Part 41
  SUPERTYPE OF (ONEOF ( length_measure_with_unit,
                        mass_measure_with_unit,
                        time_measure_with_unit,
                        electric_current_measure_with_unit,
                        thermodynamic_temperature_measure_with_unit,
                        amount_of_substance_measure_with_unit,
                        luminous_intensity_measure_with_unit,
                        plane_angle_measure_with_unit,
                        solid_angle_measure_with_unit,
                        area_measure_with_unit,
                        volume_measure_with_unit,
                        ratio_measure_with_unit ));
  value_component : measure_value;
  unit_component  : unit;
WHERE
  WR1: valid_units (SELF);
END_ENTITY;

ENTITY length_measure_with_unit  -- from Part 41
  SUBTYPE OF (measure_with_unit);
WHERE
  WR1: 'MEASURE_SCHEMA.LENGTH_UNIT' IN TYPEOF
(SELF\measure_with_unit.unit_component);
END_ENTITY;

ENTITY plane_angle_measure_with_unit  -- from Part 41
  SUBTYPE OF (measure_with_unit);
WHERE
  WR1: 'MEASURE_SCHEMA.PLANE_ANGLE_UNIT' IN
       TYPEOF (SELF\measure_with_unit.unit_component);
END_ENTITY;


ENTITY length_unit  -- from Part 41
  SUBTYPE OF (named_unit);
WHERE
  WR1:(SELF\named_unit.dimensions.length_exponent = 1.0)AND
      (SELF\named_unit.dimensions.mass_exponent = 0.0)  AND
      (SELF\named_unit.dimensions.time_exponent = 0.0)  AND
      (SELF\named_unit.dimensions.electric_current_exponent = 0.0) AND
      (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
      0.0) AND
      (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)AND
      (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
END_ENTITY;

ENTITY plane_angle_unit  -- from Part 41
  SUBTYPE OF (named_unit);
WHERE
  WR1:(SELF\named_unit.dimensions.length_exponent  = 0.0) AND
```

```
      (SELF\named_unit.dimensions.mass_exponent    = 0.0) AND
      (SELF\named_unit.dimensions.time_exponent    = 0.0) AND
      (SELF\named_unit.dimensions.electric_current_exponent = 0.0) AND
      (SELF\named_unit.dimensions.thermodynamic_temperature_exponent =
      0.0) AND
      (SELF\named_unit.dimensions.amount_of_substance_exponent = 0.0)AND
      (SELF\named_unit.dimensions.luminous_intensity_exponent = 0.0);
END_ENTITY;
```

Only length unit and plane angle unit are shown above. The subtype units each have a WHERE rule that checks that all the named unit dimensions are consistent for the unit type: zero except for length_exponent for length measures.

Given the above definitions, a length measure and unit might be:

```
#20 = LENGTH_MEASURE_WITH_UNIT (40.0, #30);
#30 = ( NAMED_UNIT(*) LENGTH_UNIT (#40) SI_UNIT ($, .METRE.) );
#40 = DIMENSIONAL_EXPONENTS (1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);
```

Length_measure_with_unit is one of the subtypes of measure_with_unit. Its two attribues, value_component and unit_component are inherited from measure_with_unit. In the named_unit, two subtypes are used. One subtype is length_unit, with the inherited exponents, and the other are the SI_units, with the SI_unit_name. The attribute type of `unit` is a SELECT of either a `named_unit` or `derived`_unit. The SELECT requires surrounding parentheses. An implication of the method used for representing measured units is that three lines of definition are required for each value stored at the instance level: for the measure type and value, for the type of unit, and one for the dimensional exponents. The easiest way to write out the values is to generate three lines for each value. However, the last two lines are likely to be the same for large sets of measurements and may be written once and referenced many times, allowing significant savings. In the latter case, the application program must maintain an in-memory listing of the instance entities that are to be shared by multiple value assignments.

An example of two different derived units and measures might be:

```
#14 = MOMENT_MEASURE_WITH_UNIT(4.7838, #5);
#5 = MOMENT_UNIT (#6, #7);
#6 = DERIVED_UNIT_ELEMENT (#3, 1.0);
#7 = DERIVED_UNIT_ELEMENT (#8, 1.0);
#3 = (NAMED_UNIT($) FORCE_UNIT() SI_UNIT(.KILO.,.NEWTON.));
#8 = (NAMED_UNIT($) LENGTH_UNIT() SI_UNIT($,.METRE.));
```

In the first case, we see that the force measure requires only a single extra line. In the second case, the moment measures require four extra lines, in addition to the value carried in instance #14. Again, these four lines may be written once and referenced multiple times by different `MOMENT_MEASURE_WITH_UNITS`.

Imperial Units

In the second meeting, the Leeds people presented how to specify "imperial units". These are specified in the Part 16c of CIS/2 (on the CD-ROM). The base units also are defined with three lines, as shown below. The main change appears to be in the `NAMED_UNIT` becoming defined as a `CONTEXT_DEPENDENT_UNIT`.

```
#22 = LENGTH_MEASURE_WITH_UNIT (240.0, #111);
#111 = (NAMED_UNIT(#10)CONTEXT_DEPENDENT_UNIT('INCH')LENGTH_UNIT());
#10= DIMENSIONAL_EXPONENTS(1.0,0.0,0.0,0.0,0.0,0.0,0.0);
```

All length measure in CIS/2 are specified in inches. The other base units are defined similarly:

```
/* area units */
#22 = MASS_MEASURE_WITH_UNIT (100.0, #221);
#221=(NAMED_UNIT(#220)CONTEXT_DEPENDENT_UNIT('POUND')MASS_UNIT());
#220= DIMENSIONAL_EXPONENTS(0.0,1.0,0.0,0.0,0.0,0.0,0.0);

/* plane angle units */
#23 = PLANE_ANGLE_MEASURE_WITH_UNIT (95.0, #551);
#551=(NAMED_UNIT(#50)CONTEXT_DEPENDENT_UNIT('DEGREE')
                                                PLANE_ANGLE_UNIT());
#50= DIMENSIONAL_EXPONENTS(0.0,0.0,0.0,0.0,0.0,0.0,0.0);

/* area units */
#24 = AREA_MEASURE_WITH_UNIT (2295.5, #771);
#771=(NAMED_UNIT(#70)AREA_UNIT()CONTEXT_DEPENDENT_UNIT('SQUARE_INCH'));
#70= DIMENSIONAL_EXPONENTS(2.0,0.0,0.0,0.0,0.0,0.0,0.0);

/* volume units */
#25 = VOLUME_MEASURE_WITH_UNIT (360.0, #881);
#881=(NAMED_UNIT(#80)CONTEXT_DEPENDENT_UNIT('CUBIC_INCH')VOLUME_UNIT());
#80= DIMENSIONAL_EXPONENTS(3.0,0.0,0.0,0.0,0.0,0.0,0.0);

/* force units */
#26 = FORCE_MEASURE_WITH_UNIT (300.0, #991);
#991=(NAMED_UNIT(#100)CONTEXT_DEPENDENT_UNIT('KIP')FORCE_UNIT());
#100= DIMENSIONAL_EXPONENTS(1.0,1.0,-2.0,0.0,0.0,0.0,0.0);
```

It should be noted that the default value of the dimensional_exponents are angle units. The derived units are specified similarly. For example, a distributed load, defined as force per unit length, has the following units definitions:

```
#14 = FORCE_PER_LENGTH_MEASURE_WITH_UNIT(4.0, #32);
#32= FORCE_PER_LENGTH_UNIT((#130,#131 ));
#130= DERIVED_UNIT_ELEMENT(#1111,1.0);
#131= DERIVED_UNIT_ELEMENT(#11,-1.0);
#1111=(NAMED_UNIT(#100)CONTEXT_DEPENDENT_UNIT('KIP')FORCE_UNIT());
#100= DIMENSIONAL_EXPONENTS(1.0,1.0,-2.0,0.0,0.0,0.0,0.0);
#11= (NAMED_UNIT(#10)CONTEXT_DEPENDENT_UNIT('INCH')LENGTH_UNIT());
#10= DIMENSIONAL_EXPONENTS(1.0,0.0,0.0,0.0,0.0,0.0,0.0);
```

Thus eight instance entities seem to be required for each derived unit. Again, the last seven can be shared for all values having the same unit.