OVERVIEW OF CIS/2
STRUCTURAL STEEL ANALYSIS MODEL
(Based on LPM500, Version from January 27, 2000)
Chuck Eastman

In this note, I try to lay out the structure of the CIS/2 EXPRESS entities and how they are used to define object instances, based on the CIS/2 schema. For each cluster of entities, generally consistent with what CIS/2 calls a base conformance class, I present the EXPRESS-G diagram, followed by the corresponding EXPRESS code, followed by the expanded set of attributes of an Entity resulting from inheritance, followed by an example set of instance entities in Part 21 file format. At each level, I provide some discussion on the intended use of the model. With this redundancy and multiple presentations, the structure should become clear from a careful reading. It is assumed that readers of this report are familiar with EXPRESS and EXPRESS-G.

These notes have been revised to reflect the LPM500 version, as dated above. It has many changes from the earlier beta releases.

This description of the CIS/2 model is expanded from those presented by Watson and Crowley. Their description filters out the relevant subset of the inheritance and attributes paths that are relevant to a part of the model. This makes interpretation easy, but does not facilitate understanding of the overall model. I have highlighted the relevant paths used to make the subsets easier to interpret.

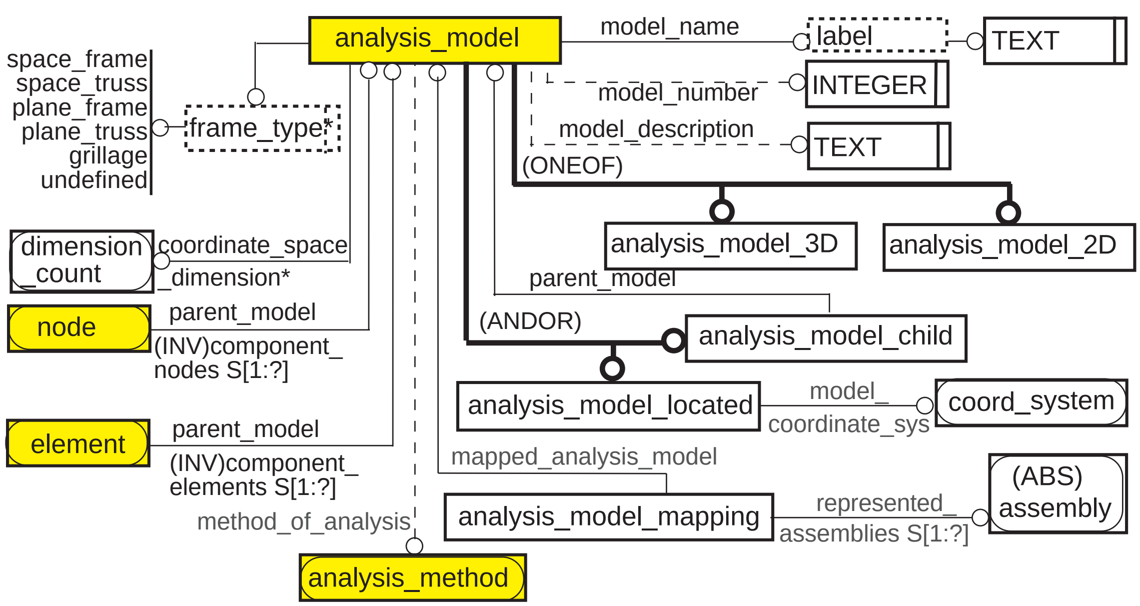

Figure One: The top level Entity definitions for an analysis model.

An overview of the EXPRESS-G Entities that typically comprise an analysis model are shown in Figure One. An analysis model begins with the definition at the top level of an analysis_model entity, carrying attributes that specify its name, model_description, a frame_type, which offers a set of enumerated alternatives, coordinate_space_dimension and optionally an analysis_method. An analysis_model may be decomposed hierarchically, with sub-models within a larger one. These are defined by using the subtype analysis_model_child. An initial analysis_model_child may be further decomposed multiple times. Each sub-model refers to its parent_model. Any of these models may have an optional location, defined as an analysis_model_located, which associates a coordinate system. Located analysis models are identified by being of a different subtype. Analaysis_model has two subtypes that may be used in its place: anaylsis_model_2D and analysis_model_3D. The analysis_model_2D has a where rule that an instance's dimension_count is 2 and its model_type is grillage, plane_truss, or plane_frame. Analysis_model_3D where rule requires that its instances have dimension_count = 3 and its model_type be space_truss or space_frame.

An analysis model also refers to and is referenced from the elements and nodes that comprise it. These are shown on the left of Figure One (and are defined in more detail below). These relations are two-way, defined through an INVERSE relation. These are the main entities making up the analysis model and provide access in the model from an analysis model and its components. Each analysis_model can be referenced by an analysis_model_mapping, which associates the analysis model with the corresponding design model (discussed elsewhere) if it exists.

The corresponding EXPRESS definitions (long form) are listed below.

```
ENTITY analysis_model
SUPERTYPE OF (ONEOF
                    (analysis_model_2D,
                    analysis_model_3D) ANDOR
                    analysis_model_located ANDOR
                    analysis_model_child);
          model_name : label;
          model_description : OPTIONAL text;
          model_type : frame_type;
          method_of_analysis : OPTIONAL analysis_method;
          coordinate_space_dimension : dimension_count;
INVERSE
          component_elements : SET [1:?] OF element
                    FOR parent_model;
          component_nodes : SET [2:?] OF node
                    FOR parent_model;

ENTITY analysis_model_2D
SUBTYPE OF (analysis_model);
WHERE
    WRA2 : SELF\analysis_model.coordinate_space_dimension = 2;
    WRA3 : (SELF\analysis_model.model_type = PLANE_FRAME) OR
        (SELF\analysis_model.model_type = PLANE_TRUSS) OR
        (SELF\analysis_model.model_type = GRILLAGE);
END_ENTITY;

ENTITY analysis_model_3D
SUBTYPE OF (analysis_model);
WHERE
    WRA4 : SELF\analysis_model.coordinate_space_dimension = 3;
    WRA5 : (SELF\analysis_model.model_type = SPACE_FRAME) OR
        (SELF\analysis_model.model_type = SPACE_TRUSS);
END_ENTITY;
```

```
ENTITY analysis_model_located
SUBTYPE OF (analysis_model);
        model_coord_sys : coord_system;
WHERE
        WRA8 : SELF\analysis_model.coordinate_space_dimension <=
                model_coord_sys.coord_system_dimensionality;
END_ENTITY;

ENTITY analysis_model_mapping;
    mapped_analysis_model : analysis_model;
    represented_assemblies : SET [1:?] OF assembly;
END_ENTITY;

ENTITY analysis_model_child
SUBTYPE OF (analysis_model);
    parent_model : analysis_model;
WHERE
    WRA6 : parent_model :<>: (SELF);
    WRA7 : SELF\analysis_model.coordinate_space_dimension <=
        parent_model.coordinate_space_dimension;
END_ENTITY;
```

If an analysis_model_3D instance expanded to include all the inherited attributes and relations, it takes the form:

```
ENTITY analysis_model_3D – expanded form
        (analysis_model:
        model_name : label;
        model_description : OPTIONAL text;
        model_type : frame_type;
        method_of_analysis : OPTIONAL analysis_method;
        coordinate_space_dimension : dimension_count;
INVERSE
        component_elements : SET [1:?] OF element
                FOR parent_model;
        component_nodes : SET [2:?] OF node
                FOR parent_model;
       );
WHERE
    WRA4 : SELF\analysis_model.coordinate_space_dimension = 3;
    WRA5 : (SELF\analysis_model.model_type = SPACE_FRAME) OR
        (SELF\analysis_model.model_type = SPACE_TRUSS);
END_ENTITY;
```

A corresponding Part 21 file might be:

```
#1 analysis_model_3D('CIS2 test structure','example',.SPACE_FRAME.,
#2,3);
```

analysis_model_3D is the analysis model subtype, 'CIS2 test structure' is the model name, the optional model_description is 'example', .SPACE_FRAME. is one of the enumerated frame_types, method_of_analysis is #2 below and 3 is the dimension_count of the model. Notice that in a Part 21 file, the inverse relations are ignored. A simple example also can ignore the analysis_model subtypes, the analysis_model_mapping and analysis_model_child. Analysis_model_located can be a shared supertype with any of these, adding a coordinate system attribute. The analysis_model_3D applies some WHERE clauses to verify the dimension_count and frame_type.

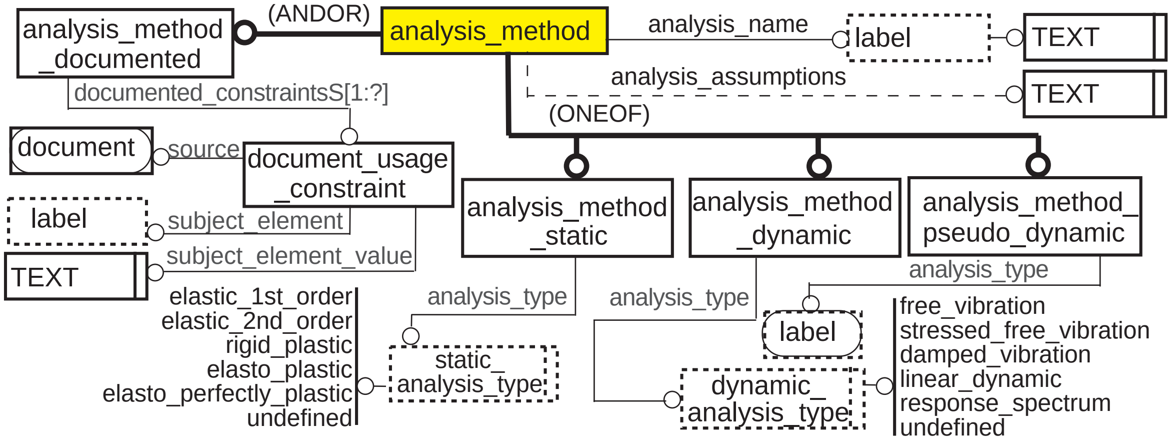

Figure Two: EXPRESS-G depiction of analysis_method.

An optional attribute of analysis_model is analysis_method, shown in Figure Two. It specifies the name of the analysis and carries text regarding any assumptions. It is useful as a record of an analysis run. The analysis_method may be specialized into one of three subtypes, designating a static analysis, a pseudo-dynamic analysis or a full dynamic analysis. Of course, the analysis type may be applied to any decomposed part of the overall structure.

The EXPRESS definitions of analysis_method and its subtypes follow.

```
ENTITY analysis_method
SUPERTYPE OF (ONEOF
        (analysis_method_dynamic,
        analysis_method_pseudo_dynamic,
        analysis_method_static) ANDOR
        analysis_method_documented);
    analysis_name : label;
    analysis_assumptions : OPTIONAL text;
END_ENTITY;

ENTITY analysis_method_documented
SUBTYPE OF (analysis_method);
    documented_constraints : SET [1:?] OF document_usage_constraint;
END_ENTITY;

ENTITY analysis_method_dynamic
SUBTYPE OF (analysis_method);
    analysis_type : dynamic_analysis_type;
END_ENTITY;

ENTITY analysis_method_pseudo_dynamic
SUBTYPE OF (analysis_method);
    analysis_type : label;
END_ENTITY;
```

```
ENTITY analysis_method_static
SUBTYPE OF (analysis_method);
        analysis_type : static_analysis_type;
END_ENTITY;

ENTITY document_usage_constraint;  --from Part 41
        source : document;
        subject_element : label;
        subject_element_value : text;
END_ENTITY;  --  STEP Part 41
```

If we flatten the inheritance to analysis_method_static, we get:

```
        ENTITY analysis_method_static
         (analysis_method:
        analysis_name : label;
         analysis_assumptions : OPTIONAL text;
        )
         analysis_type : static_analysis_type;
        END_ENTITY;
```

The Part 21 definition of analysis_method in a simple form might be:

```
#2= ANALYSIS_METHOD_STATIC ('standard analysis',$,'elastic_1st_order');
```

analysis_method_static is the subtype used, with the analysis_name given and analysis_assumptions left blank because it is optional. An attribute of static_analysis_method is analysis_type, which is here given as 'elastic_first_order', taken from the enumerated set.

NODES

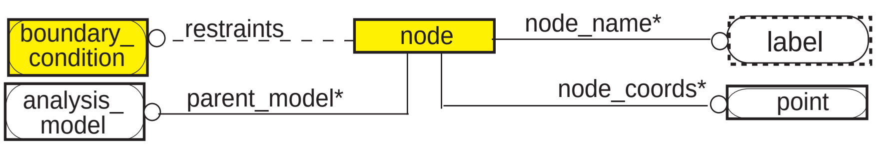

Figure Three: The CIS/2 definition of a node.

The key elements of an analysis model are the nodes and elements. The EXPRESS-G diagram of the node entity is show in Figure Three. It has a name and a number, references the analysis_model it is part of and its boundary conditions. Its geometrical location is defined by a cartesian_point, an entity adopted from the STEP Part 042 integrated resources. Two UNIQUE rules require that the node name and the node coordinates for a given parent model be unique. A WHERE clause requires that the dimensionality of a node be the same as its parent model.

The node EXPRESS definition follows.

```
ENTITY node;
        node_name : label;
        node_coords : point;
        restraints : OPTIONAL boundary_condition;
```

```
        parent_model : analysis_model;
UNIQUE
        URN1 : node_name, parent_model;
        URN2 : node_coords, parent_model;
WHERE
        WRN1 : node_coords.dim = parent_model.coordinate_space_dimension;
END_ENTITY;
```

The node has no inheritances. Example part 21 file entries for node are:

```
#9=NODE ('N1', #10, #60, #1);
#10=NODE ('N2', #11, $, #1);
```

where in the first node, 'N1' is the node_name, #10 refers to the node_coords, #60 refers to the restraint (which is optional) and #1 refers to the parent_model it is part of. $ indicates a null value. Data is required unless it is specifically identified as optional (shown as a dotted line in EXPRESS-G).

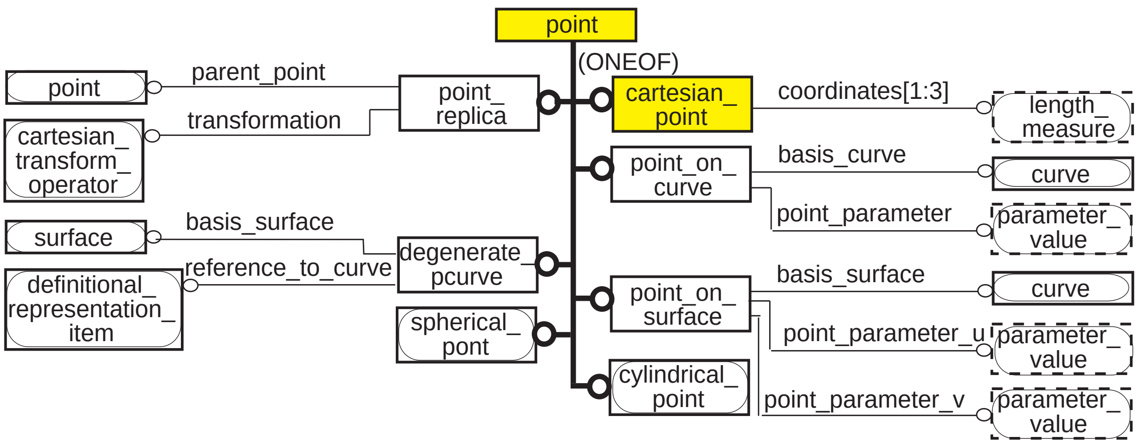

Figure Four: The point entity and some of its subtypes as defined in the STEP Part 042 integrated resources.

Cartesian_ point is a subtype of point, defined in the STEP Part 42 Integrated Resources. Most of the subtypes of point are shown in Figure Four. The main one, cartesian_ point is defined with three length measures. It is the main point entity used in CIS/2, although point_on_curve is also used. Point is a subtype of geometrical_representation_item, defined in the General Issues of Representation Section.

```
ENTITY point      -- expanded STEP Part 42
SUPERTYPE OF (ONEOF(
    cartesian_point,
    point_on_curve,
    point_on_surface,
    point_replica,
    degenerate_pcurve,
    cylindrical_point,
    spherical_point))
SUBTYPE OF(geometric_representation_item);
END_ENTITY;

ENTITY cartesian_point  -- STEP Part 42
SUBTYPE OF (POINT);
```

```
   coordinates ; LIST [1:3] OF length_measure;
END_ENTITY;

ENTITY point_on_curve  -- STEP Part 42
SUBTYPE OF(point);
    basis_curve : curve;
    point_parameter : parameter_value;
END_ENTITY;

ENTITY point_on_surface  -- STEP Part 42
SUBTYPE OF(point);
    basis_surface : surface;
    point_parameter_u : parameter_value;
    point_parameter_v : parameter_value;
END_ENTITY;

ENTITY point_replica  -- STEP Part 42
SUBTYPE OF(point);
    parent_pt: point;
    transformation : cartesian_transformation_operator;
WHERE
    WR42A1 : transformation.dim = parent_pt.dim;
    WR42B2 : acyclic_point_replica (SELF,parent_pt);
END_ENTITY;

ENTITY cylindrical_point   -- STEP Part 42
SUBTYPE OF(point);
        r_coordinate : length_measure;
        theta_coordinate : plane_angle_measure;
        z_coordinate : length_measure;
END_ENTITY;

ENTITY spherical_point -- STEP Part 42
SUBTYPE OF(point);
        r_coordinate : length_measure;
        phi_coordinate : plane_angle_measure;
        theta_coordinate : plane_angle_measure;
END_ENTITY;
```

For simple structures, Cartesian_point will almost always be used. Point_on_curve or point_on_surface may be needed in special conditions.

The corresponding Part 21 file entries for cartesian_point is:

```
#10=CARTESIAN_POINT ('NODE N1',(0.0, 0.0, 0.0));
#11=CARTESIAN_POINT ('NODE N2',(120.0, 0.0, 0.0));
```

where 'NODE N1' is the name inherited from representation_item (see Resentations and Measurements tutorial) and the numbers in parentheses are the three length_measures of cartesian_coordinate.

An important property of a node is its boundary conditions. These are defined in Figure Five. It begins with an abstract type that defines three angle measures that adjust the x-, y- and z-alignments of the axes. It also provides a name, and optionally a description for each boundary condition. A boundary condition may be logical, linear or non-linear. Logical boundaries are 100 percent rigid or free. Linear boundary conditions consist of stiffness measures for the three displacements and three rotations about the joint. It also includes a warping stiffness measure. The non-linear boundary condition is a list of values that alter the linear conditions.

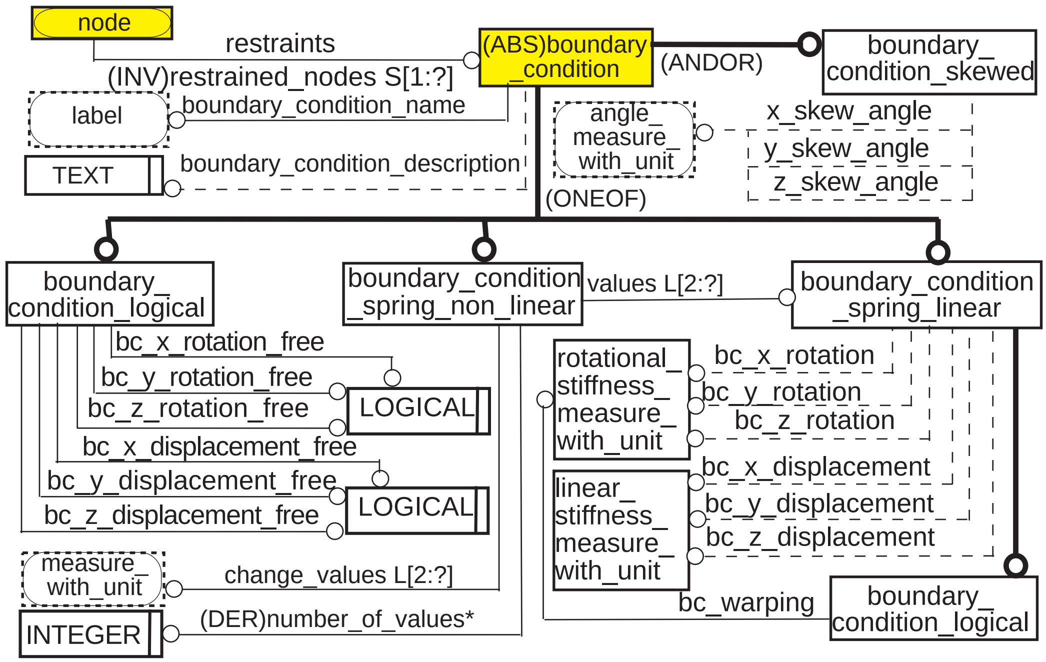

Figure Five: The definition of boundary conditions in CIS/2.

The EXPRESS definitions for boundary condition is below.

```
ENTITY boundary_condition
ABSTRACT SUPERTYPE OF (ONEOF
    (boundary_condition_logical, boundary_condition_spring_linear,
    boundary_condition_spring_non_linear) ANDOR boundary_condition_skewed);
    boundary_condition_name : label;
    boundary_condition_description : OPTIONAL text;
INVERSE
    restrained_nodes : SET [1:?] OF node FOR restraints;
END_ENTITY;

ENTITY boundary_condition_logical
SUBTYPE OF (boundary_condition);
    bc_x_displacement_free : LOGICAL;
    bc_y_displacement_free : LOGICAL;
    bc_z_displacement_free : LOGICAL;
    bc_x_rotation_free : LOGICAL;
    bc_y_rotation_free : LOGICAL;
    bc_z_rotation_free : LOGICAL;
END_ENTITY;

ENTITY boundary_condition_spring_linear
SUBTYPE OF (boundary_condition);
    bc_x_displacement : OPTIONAL linear_stiffness_measure_with_unit;
    bc_y_displacement : OPTIONAL linear_stiffness_measure_with_unit;
    bc_z_displacement : OPTIONAL linear_stiffness_measure_with_unit;
```

```
    bc_x_rotation : OPTIONAL rotational_stiffness_measure_with_unit;
    bc_y_rotation : OPTIONAL rotational_stiffness_measure_with_unit;
    bc_z_rotation : OPTIONAL rotational_stiffness_measure_with_unit;
WHERE
    WRB10 : EXISTS (bc_x_displacement) OR EXISTS (bc_y_displacement) OR EXISTS
(bc_z_displacement);
    WRB11 : EXISTS (bc_x_rotation) OR EXISTS  (bc_y_rotation) OR EXISTS (bc_z_rotation);
END_ENTITY;

ENTITY boundary_condition_spring_non_linear
SUBTYPE OF (boundary_condition);
        change_values : LIST [2:?] OF measure_with_unit;
        values : LIST [2:?] OF boundary_condition_spring_linear;
DERIVE
        number_of_values : INTEGER := SIZEOF(change_values);
WHERE
        WRB12 : SIZEOF(values) = SIZEOF(change_values);
END_ENTITY;

ENTITY boundary_condition_warping
SUBTYPE OF (boundary_condition_spring_linear);
        bc_warping : rotational_stiffness_measure_with_unit;
END_ENTITY;
```

All boundary_conditions inherit a name and description. The boundary_condition_logical allows simple logical definition of releases. The full definition of some of these elements requires inclusion of units of measurement. These are not defined here but are presented in the General Representation Issues tutorial, which addresses all units and measures.

Two different Part 21 file entries for boundary conditions follow. First is the LOGICAL form, followed by the longer numerical form.

```
#51=boundary_condition_logical('roller_x','$',
                                          .T.,.T.,.T.,.T.,.T.,.T.);
#52=boundary_condition_logical('pinned_yz','$',
                                          .F.,.F.,.F.,.F.,.T.,.T.);

#60=boundry_condition_spring_linear( 1, 'PINNED',
      'test boundary condition', #110, #110, #110, #115, #115, #115);
#110=linear_stiffness_measure_with_unit( -1.0, #1003);
#115=rotational_stiffness_measure_with unit( -1.0, #1005);
#1003=linear_stiffness_unit((#1111, #1112)); -- force, length
#1005=rotational_stiffness_unit((#1111, #1112, #1132));
      -- force, length, plane angle
#1111 = force_measure_with_unit(61.1639, #300);
#1112 = length_measure_with_unit (125.00, #400);
#1113 = angle_measure_with_unit (1.570, #500);
#300 = (force_unit() named_unit($)si_unit(.KILO.,.NEWTON.));
#400=(length_unit()named_unit($)si_unit($,.METRE.));
#500 = (angle_unit() named_unit($)si_unit($,.RADIAN.));
```

Since boundary_condition is abstract, instances can only be made of its subtypes. Here, boundary_condition_logical is used first, then boundary_condition_spring_linear is used. Its inherited attributes are a name and description This is followed with three linear_stiffness_ measure_with_units and three rotational_stiffness_measure_with_units. Linear stiffness is defined as a force transmitted over a length and is a derived unit composed of these three entities.

Rotational_stiffness_measure_with_unit is another derived_unit composed of a force, length and plane angle. (These measure derivations have not yet been tested.)

ELEMENTS

The elements that are composed with nodes in an analysis model are defined in Figure Six. They are one of four subtypes of different dimensionality: a point, a curve, a surface or a volume. They also may be combined using ANDOR to include material. An element is defined by a name, description, dimensionality and reference to the analysis_model it is part of. The dimensionality must be consistent with what subtype it is.

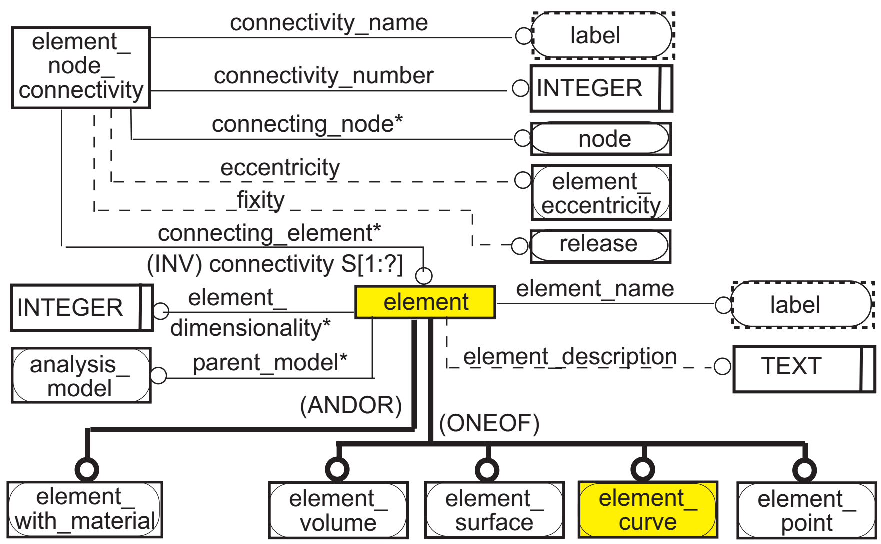

Figure Six: The EXPRESS-G description of element and element node connectivity.

Both elements and nodes can be defined, then their connectivity specified, using element_node_connectivity. Figure Six depicts how element_node_connectivity (ENC) relates to nodes and elements. There is one or more ENC for each element. Because an element may be a point-element, a line-element, a sheet-element or 3-D shape element, elements have different numbers of connections to nodes. Typically, a point element has one connection, a linear element has two, a sheet element has three or more and a volume element has four or more. Each element_node_connectivity entity has a name, a number and optional label and the node it connects to. It also has an optional release and element_eccentricity.

The corresponding EXPRESS definitions are below.

```
ENTITY element
SUPERTYPE OF (ONEOF(element_volume, element_surface, element_curve, element_point)
ANDOR element_with_material);
    element_name : label;
    element_description : OPTIONAL text;
    parent_model : analysis_model;
    element_dimensionality : INTEGER;
INVERSE
    connectivity : SET [1:?] OF element_node_connectivity FOR connecting_element;
UNIQUE
    URE1 : element_name, parent_model;
WHERE
```

```
    WRE2 : element_dimensionality <= parent_model.coordinate_space_dimension;
END_ENTITY;
```

After the identifiers, element references the parent analysis model. Its dimensionality follows, defined as (point, linear, plate, solid corresponding to 0,1,2,3 respectively).

```
ENTITY element_node_connectivity;
    connectivity_number : INTEGER;
    connectivity_name : label;
    connecting_node : node;
    connecting_element : element;
    eccentricity : OPTIONAL element_eccentricity;
    fixity : OPTIONAL release;
UNIQUE
    URE2 : connecting_node, connecting_element;
WHERE
    WRE9 : NOT( (connectivity_number > 2) AND (connecting_element.element_dimensionality <
2) );
    WRE10 : NOT( (connectivity_name <> 'Start Node') AND (connectivity_number = 1) );
    WRE11 : NOT( (connectivity_name <> 'End Node') AND (connectivity_number = 2) AND
(connecting_element.element_dimensionality = 1) );
    WRE12 : connecting_node.parent_model :=: connecting_element.parent_model;
END_ENTITY;
```

Element_node_connectivity has two identifiers—name and number— where number =1 corresponds to 'start_node' and number = 2 corresponds to 'end node'. It references the single node and single element it connects. Each also specifies an associated release. The newest release has added a number of WHERE clauses, constraining possible values. These do the following checking:

WRE9: a one or two dimensional element cannot have a connectivity number greater than 2
WRE10: tells if connectivity name is 'start node' and connectivity_number is 1
WRE11: tells if connectivity name is 'end node' and connectivity_number is 2
WRE12: checks that the parent model for connectinig_element and connecting_node are consistent.

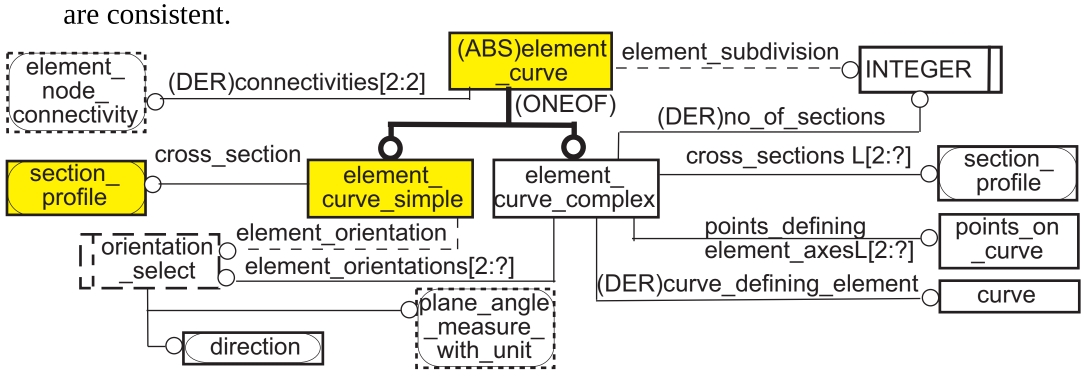

Figure Seven: The definition of element_curve.

The most common element for building structures is element_curve. It is defined in EXPRESS-G in Figure Seven. In the general case, an element may be bent and may have a changing cross-section, but in the simple case, the element is linear with a constant cross-section. It has a section_profile and an orientation. Orientation_select is either a single angle or a list of angle ratios. Element_curve_complex supports multiple cross-sections, curved elements defined by points on a curve, and twisted elements having changed orientation. Element_curve and its subtypes are defined in EXPRESS below. Section_profile and its other subtypes are defined in the Section Profile tutorial.

```
ENTITY element_curve
ABSTRACT SUPERTYPE OF (ONEOF(element_curve_simple, element_curve_complex))
SUBTYPE OF (element);
        element_subdivision : OPTIONAL INTEGER;
DERIVE
        connectivities : SET [2:2] OF element_node_connectivity := bag_to_set
        (USEDIN(SELF, STRUCTURAL_FRAME_SCHEMA.ELEMENT_NODE_
        CONNECTIVITY. CONNECTING_ ELEMENT'));
WHERE
        WRE3 : SELF\element.element_dimensionality = 1;
        WRE4 : connectivities[1] :<>: connectivities[2];
        WRE5 : connectivities[1].connecting_node :<>: connectivities[2].connecting_node;
END_ENTITY;

ENTITY element_curve_complex
SUBTYPE OF (element_curve);
        cross_sections : LIST [2:?] OF section_profile;
        points_defining_element_axis : LIST [2:?] OF point_on_curve;
        element_orientations : LIST [2:?] OF orientation_select;
DERIVE
        number_of_sections : INTEGER := SIZEOF (cross_sections);
        curve_defining_element : curve :=
                points_defining_element_axis[1]\point_on_curve.basis_curve;
WHERE
        WRE6 : ( (SIZEOF (points_defining_element_axis) = number_of_sections) AND
                        (SIZEOF (element_orientations) = number_of_sections) );
        WRE7 : SIZEOF(QUERY(temp <* points_defining_element_axis |
                (temp\point_on_curve.basis_curve) :<>: curve_defining_element)) = 0;
END_ENTITY;

ENTITY element_curve_simple
SUBTYPE OF (element_curve);
        cross_section : section_profile;
        element_orientation : orientation_select;
END_ENTITY;

TYPE orientation_select
 = SELECT
        (plane_angle_measure_with_unit, direction);
END_TYPE;
```

The WHERE clauses for element_curve check:

WRE3: that the element_dimensionality of the element is 1

WRE4: that the first two connectivities are not to the same element_ node_connectivity

WRE5: that the first two nodes being connected are not identical

The orientation of the element_curve_simple may be defined with a direction or angle measure.

If we collapse element into element_curve into element_curve_simple, we get the following entity structure:

```
ENTITY element_curve_simple
    (element
        element_name : label;
        element_description : OPTIONAL text;
        parent_model : analysis_model;
        element_dimensionality : INTEGER;
    INVERSE
        connectivity : SET [1:?] OF element_node_connectivity FOR connecting_element;
    UNIQUE
        URE1 : element_name, parent_model;
    WHERE
        WRE2 : element_dimensionality <= parent_model.coordinate_space_dimension;
     );
     (element_curve:
    element_subdivision : OPTIONAL INTEGER;
    DERIVE
                connectivities : SET [2:2] OF element_node_connectivity := bag_to_set
                (USEDIN(SELF,'STRUCTURAL_FRAME_SCHEMA.ELEMENT_NODE_
                CONNECTIVITY.CONNECTING_ ELEMENT'));
   WHERE
          WRE3 : SELF\element.element_dimensionality = 1;
          WRE4 : connectivities[1] :<>: connectivities[2];
          WRE5 : connectivities[1].connecting_node :<>: connectivities[2].connecting_node;
   );
          cross_section : section_profile;
          element_orientation : orientation_select;
END_ENTITY;
```

Instance definitions of elements and element_node_connectivity follows.

```
#140 = ELEMENT_CURVE_SIMPLE ('column',$,#1,2,$,#1102,#100);
#250 = ELEMENT_NODE_CONNECTIVITY (1,'start', #9, #140,$,#43);
#251 = ELEMENT_NODE_CONNECTIVITY (2,'end', #10, #140,$,#43);
#100 = PLANE_ANGLE_MEASURE_WITH_UNIT( 0.0, #200)
#200 = (NAMED_UNIT(*)PLANE_ANGLE_UNIT()SI_UNIT($,.RADIAN.));
```

Instance #140 of element_curve_simple inherits from element a name, an optional description, parent model and dimensionality. Here the dimensionality indicates a linear element (which is required if it is an element_curve). From element_curve, it inherits an optional element_subdivision (which is set to NULL). Element_curve_simple has its own attributes: a section_profile and an orientation. The section_profile is #1102. Orientation is defined as a plane_angle_with_unit, defined in radians.

The two element_node_connectivity are identified by a connectivity_number and label and refer to nodes #9 and #10 respectively. They define the two ends of the element_curve_simple, #140. In the two element_node_connectivity instances, the element_eccentricity have been defaulted to NULL. The optional release definition is #43 below.

Releases

The release associated with element_node_connectivity is described in Figure Eight. It is much like boundary_condition. Release is an abstract type which defines certain attributes that are carried in a release_linear or release_non_linear. The attributes include a release number, description and label. It also carries an Inverse relation with the element_node_connectivity which it describes. The linear and non-linear releases are defined similarly to the bounday_conditions. The non-linear measures are defined as ratios to the linear measures.

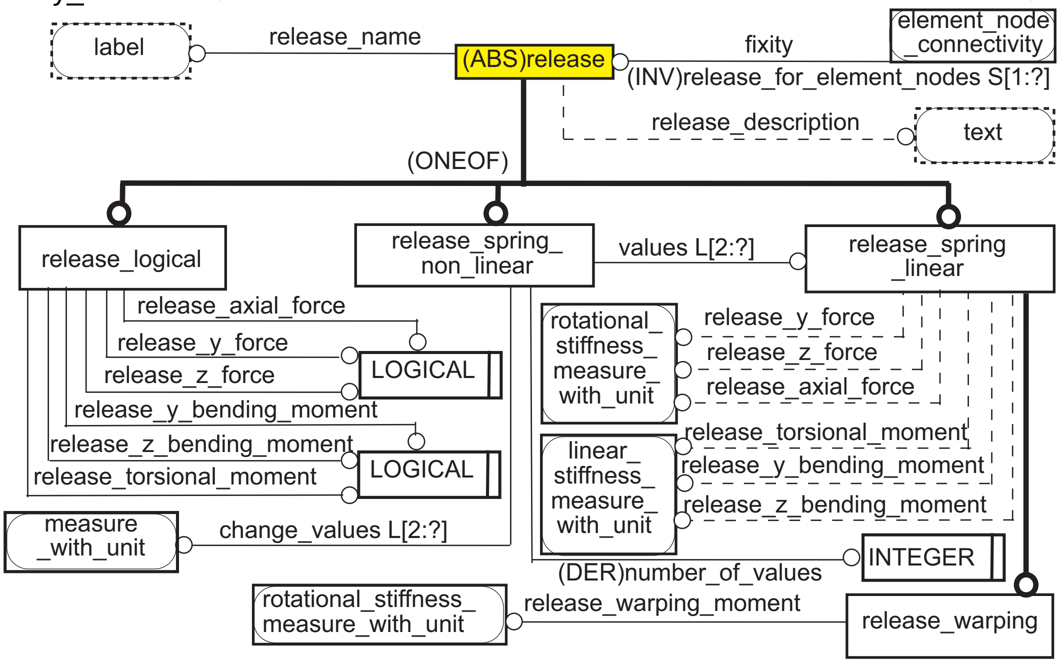

Figure Eight: The definition of an element_node_connectivity's *release.*

The EXPRESS release entity is defined below.

```
ENTITY release
ABSTRACT SUPERTYPE OF (ONEOF
        (release_logical, release_spring_linear, release_spring_non_linear));
    release_name : label;
    release_description : OPTIONAL text;
INVERSE
    release_for_element_nodes : SET [1:?] OF element_node_connectivity FOR fixity;
END_ENTITY;

ENTITY release_logical
SUBTYPE OF (release);
    release_axial_force : LOGICAL;
    release_y_force :  LOGICAL;
    release_z_force : LOGICAL;
    release_torsional_moment : LOGICAL;
    release_y_bending_moment : LOGICAL;
    release_z_bending_moment : LOGICAL;
END_ENTITY;

ENTITY release_spring_linear
SUPERTYPE OF (release_warping)
SUBTYPE OF (release);
    release_axial_force : OPTIONAL linear_stiffness_measure_with_unit;
    release_y_force :  OPTIONAL linear_stiffness_measure_with_unit;
    release_z_force : OPTIONAL linear_stiffness_measure_with_unit;
    release_torsional_moment : OPTIONAL rotational_stiffness_measure_with_unit;
    release_y_bending_moment : OPTIONAL rotational_stiffness_measure_with_unit;
```

```
    release_z_bending_moment : OPTIONAL rotational_stiffness_measure_with_unit;
WHERE
    WRR21 : EXISTS (release_axial_force) OR EXISTS (release_y_force) OR EXISTS
(release_z_force);
    WRR22 : EXISTS (release_torsional_moment) OR EXISTS (release_y_bending_moment) OR
EXISTS (release_z_bending_moment);
END_ENTITY;

ENTITY release_spring_non_linear
SUBTYPE OF (release);
        change_values : LIST [2:?] OF measure_with_unit;
        values : LIST [2:?] OF release_spring_linear;
DERIVE
        number_of_values : INTEGER := SIZEOF(change_values);
WHERE
        WRR23 : SIZEOF(values) = SIZEOF(change_values);
END_ENTITY;

ENTITY release_warping
SUBTYPE OF (release_spring_linear);
    release_warping_moment : rotational_stiffness_measure_with_unit;
END_ENTITY;
```

Some of the releases have some WHERE rules. Release_spring_linear requires at least one linear force and one bending force. Release_spring_nonlinear requires that the same number of change values exist as their cardinality count.

Release_logical is likely to be the most commonly used. If flattened to include all inherited attributes, it has the following structure:

```
ENTITY release_logical
(release:
      release_name : label;
      release_description : OPTIONAL text;
    INVERSE
      release_for_element_nodes : SET [1:?] OF element_node_connectivity FOR fixity;
    );
    release_axial_force : LOGICAL;
    release_y_force :  LOGICAL;
    release_z_force : LOGICAL;
    release_torsional_moment : LOGICAL;
    release_y_bending_moment : LOGICAL;
    release_z_bending_moment : LOGICAL;
END_ENTITY;
```

A corresponding Part 21 instance example of a two releases follow.

```
#42 = RELEASE_LOGICAL ('pinned in x-axis',$,.F.,.F.,.F.,.F.,.T., .T.);
```

Instance #42 shows release_logical. It carries a name and optional description, followed by six logical values, for the three axial forces and three rotations. Instance #42 assumes that a release_spring_logical condition being .T. indicates that IS released. It indicates a pinned joint, with the pin in the x-axis.

More complex releases involve different measures and derived_ units leading to a more complex structure. A couple of examples are generated below.

```
#43 = RELEASE_SPRING_LINEAR ('FIXED END', $, #110, #110, #110, #110,
#115, #115);
#110=LINEAR_STIFFNESS_MEASURE_WITH_UNIT(1.0, #1003);
#1003=LINEAR_STIFFNESS_UNIT((#1111, #1112)); -- force, length
#115=ROTATIONAL_STIFFNESS_MEASURE_WITH UNIT( 0.0, #1005);
#1005=ROTATIONAL_STIFFNESS_UNIT((#1111, #1112, #1113));
#1111 = FORCE_UNIT(#81);
#81= DIMENSIONAL_EXPONENTS(1.0,1.0,-2.0,0.0,0.0,0.0,0.0);
#1112 = LENGTH_UNIT (#82);
#82= DIMENSIONAL_EXPONENTS(1.0,0.0,0.0,0.0,0.0,0.0,0.0);
#1113 = PLANE_ANGLE_UNIT (#83);
#83= DIMENSIONAL_EXPONENTS(0.0,0.0,0.0,0.0,0.0,0.0,0.0);
```

A release_spring_linear moment transferring release is defined in instance #43. It also carries a name and optional description, followed by three linear stiffness measurements and three rotational stiffness measurements. Linear_stiffness_unit has WHERE rules requiring a force_unit and a length_unit for its proper definition. The units for these are defined, supported by their dimensional exponents. Rotational_stiffness_measure_with_unit is also defined. It has the constraint that its measure must be a rotational_stiffness_unit. All the dimensional_exponents are defined and constrained in their associated units.

This is a walk-through of a simple analysis structure. The loads applied and the section profiles are dealt with in separate tutorials.