ISO TC 184/SC4 WG3 N914
2000-05-04

INTERNATIONAL STANDARD ISO 10303-203:1994
AMENDMENT 1

Industrial automation systems and integration — Product data representation and exchange — Part 203: Application Protocol: Configuration controlled 3D designs of mechanical Parts and assemblies

AMENDMENT 1

Amendment 1 to International Standard ISO 10303-203:1994 was prepared by Technical Committee ISO/TC 184, *Industrial automation systems and integration*, Subcommittee SC 4, *Industrial data*.

Included SEDS reports:1, 3, 4, 6, 8, 24, 25, 26, 27, 37, 55, 56, 57, 69, 89, 117, 244, 245, 246, 316, 317, 318, 329, 330, 331, 332, 354, 355, 356, 358, 359, 360, 361, 362, 363 and 389

Introduction

This document amends ISO 10303-203:1994, Product data representation and exchange - Part 203: Application Protocol: Configuration controlled 3d designs of mechanical parts and assemblies. The corrected document supercedes ISO 10303-203:1994.

The purpose of the modifications to the text of ISO 10303-203:1994 is to correct errors in the EXPRESS definitions likely to cause compilation problems and to replace the object identifier for the document and the applicable schema.

© ISO 2000
All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

International Organization for Standardization
Case Postale 56 - CH-2111 Geneve 20 - Switzerland

Printed in Switzerland

Modifications to the text of ISO 10303-203:1994

Clause 2, p. 3, clause 5.2, pp. 89-242, annex A, pp. 270-400, table of contents, pp. iii-iv and index, pp. 482-515

The purpose of these modifications is to correct Express errors that were not detected at the time of publication due to the immaturity of the checking tools at that time. The corrections have been documented in change pages for readability. The following pages replace Clause 2, Clause 5.2, Annex A and the index of ISO 10303-203:1994 in their entirety as well as pages iii and iv of the table of contents. The net affect of these changes in the short listing is:

— *corrected geometric_representation_item_3d rule*
— *corrected assembly shape is defined function*
— *corrected subtype_mandatory_shape_representation rule*
— *corrected cc_design_date_time_correlation function*
— *added USEs from AICs and removed the OLD 203 entities and functions*
— *added dummy_gri constant*
— *added dummy_tri constant*
— *added degenerate_toroidal_surface (per TC2)*
— *added approval_person_organization_constraints (to solve type pruning issue)*
— *added approval_date_time_constraints (to solve type pruning issue)*
— *removed cartesian_transformation_opertor_2d (per tc2)*
— *removed offset_curve_2d (per tc2)*
— *added property_definition_representation (per seds 57)*
— *added representation_relationship (per seds 57)*
— *added b_spline_curve (per seds 57)*
— *added b_spline_surface (per seds 57)*
— *added conic (per seds 57)*
— *added point_replica (per seds 57)*
— *added swept_surface (per seds 57)*
— *added oriented_closed_shell (per seds 57)*
— *added path (per seds 57)*
— *added brep_with_voids (per seds 57)*

Since the long listing is generated from the short listing, the best method of presenting the long listing is to republish it as change pages. The index is being reissued to aid in navigation of the revised short and long listings. The clause 2 changes were made so the document references the appropriate 500 series parts.

4.1.11 part_identification 14
4.1.12 shape 14
4.1.13 source_control 14
4.1.14 wireframe_with_topology 15
4.2 Application objects 15
4.3 Application assertions 33

5 Application interpreted model 39
5.1 Mapping table 39
5.2 AIM EXPRESS short listing 89
5.2.1 Fundamental concepts and assumptions 97
5.2.2 Configuration controlled design constants 100
5.2.2 Configuration controlled design types 101
5.2.3 Configuration controlled design entities 105
5.2.3 Configuration controlled design imported entity modifications 116
5.2.4 Configuration controlled design rules 131
5.2.5 Configuration controlled design functions 194

6 Conformance requirements 243
6.1 Conformance class 1 entities 244
6.2 Conformance class 1b entities 245
6.3 Conformance class 2 entities 247
6.4 Conformance class 3 entities 250
6.5 Conformance class 4 entities 251
6.6 Conformance class 5 entities 253
6.7 Conformance class 6 entities 254

Annexes

A AIM EXPRESS expanded listing 270

B AIM short names of entities 401

C Protocol Implementation Conformance Statement (PICS) proforma 412

D Implementation method specific requirements 414

E Information object registration 415
E.1 Document identification 415
E.2 Schema identification 415
E.2.1 config_control_design expanded schema 415
E.2.2 config_control_design short form schema 415

F Application activity model 416
F.1 AAM definitions 417
F.2 AAM Diagrams 420

G Application reference model 423

H AIM EXPRESS-G 431

J AIM EXPRESS listing 471

K Application protocol usage guide 472
K.1 Usage test purposes 472
K.2 Example part 476

L Bibliography 481

Index 482

Figures

Figure F.1 - IDEF0 basic notation 416
Figure F.2 - A0 Manage product development in IDEF0 421
Figure F.3 - A3 Develop product design in IDEF0 422
Figure G.1 - ARM diagram 1 of 7 in IDEF1X 424
Figure G.2 - ARM diagram 2 of 7 in IDEF1X 425
Figure G.3 - ARM diagram 3 of 7 in IDEF1X 426
Figure G.4 - ARM diagram 4 of 7 in IDEF1X 427
Figure G.5 - ARM diagram 5 of 7 in IDEF1X 428
Figure G.6 - ARM diagram 6 of 7 in IDEF1X 429
Figure G.7 - ARM diagram 7 of 7 in IDEF1X 430
Figure H.1 - application context - AIM EXPRESS-G diagram 1 of 39 432
Figure H.2 - product definition - AIM EXPRESS-G diagram 2 of 39 433
Figure H.3 - product category - AIM EXPRESS-G diagram 3 of 39 434
Figure H.4 - property definition - AIM EXPRESS-G diagram 4 of 39 435
Figure H.5 - property representation - AIM EXPRESS-G diagram 5 of 39 436
Figure H.6 - shape representation relationship - AIM EXPRESS-G diagram 6 of 39 437
Figure H.7 - representation - AIM EXPRESS-G diagram 7 of 39 438
Figure H.8 - geometric representation items - AIM EXPRESS-G diagram 8 of 39 439
Figure H.9 - topological representation items AIM EXPRESS-G diagram 9 of 39 440
Figure H.10 - point - AIM EXPRESS-G diagram 10 of 39 441
Figure H.11 - geometric orientation - AIM EXPRESS-G diagram 11 of 39 442
Figure H.12 - curve - AIM EXPRESS-G diagram 12 of 39 443
Figure H.13 - conic - AIM EXPRESS-G diagram 13 of 39 444
Figure H.14 - bounded curves - AIM EXPRESS-G diagram 14 of 39 445
Figure H.15 - surface curve - AIM EXPRESS-G diagram 15 of 39 446
Figure H.16 - b-spline curve - AIM EXPRESS-G diagram 16 of 39 447
Figure H.17 - surface - AIM EXPRESS-G diagram 17 of 39 448

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this International Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this International Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of the IEC and ISO maintain registers of currently valid International Standards.

ISO 31:1992, *Quantities and units*.

ISO 1000:1992, *SI units and recommendations for the use of their multiples and of certain other units*.

ISO/IEC 8824-1:–[1)], *Information technology - Open systems interconnection - Abstract syntax notation one (ASN.1) - Part 1: Specification of basic notation*.

ISO 10303-1:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 1: Overview and fundamental principles*.

ISO 10303-11:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual*.

ISO 10303-21:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 21: Clear text encoding of the exchange structure*.

ISO 10303-31:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 31: Conformance testing methodology and framework: General concepts*.

ISO 10303-41:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 41: Integrated generic resources: Fundamentals of product description and support*.

ISO 10303-42:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 42: Integrated generic resources: Geometric and topological representation*.

ISO 10303-43:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 43: Integrated generic resources: Representation structures*.

ISO 10303-44:1994, *Industrial automation systems and integration - Product data representation and exchange - Part 44: Integrated generic resources: Product structure configuration*.

[1)] To be published.

ISO 10303-501, *Industrial automation systems and integration | Product data representation and exchange | Part 501: Application interpreted construct: Edge-based wireframe.*

ISO 10303-502, *Industrial automation systems and integration | Product data representation and exchange | Part 502: Application interpreted construct: Shell-based wireframe.*

ISO 10303-507, *Industrial automation systems and integration - Product data representation and exchange | Part 507: Application interpreted construct: Geometrically bounded surface.*

ISO 10303-509, *Industrial automation systems and integration - Product data representation and exchange | Part 509: Application interpreted construct: Manifold surface.*

ISO 10303-510, *Industrial automation systems and integration - Product data representation and exchange | Part 510: Application interpreted construct: Geometrically bounded wireframe.*

ISO 10303-511, *Industrial automation systems and integration - Product data representation and exchange | Part 511: Application interpreted construct: Topologically bounded surface.*

ISO 10303-512, *Industrial automation systems and integration - Product data representation and exchange | Part 512: Application interpreted construct: Faceted boundary representation.*

ISO 10303-514, *Industrial automation systems and integration - Product data representation and exchange | Part 514: Application interpreted construct: Advanced boundary representation.*

This Page Intentionally Left Blank

3 Definitions and abbreviations

For the purposes of this part of ISO 10303, the following definitions and abbreviations apply.

3.1 Terms defined in ISO 10303-1

This part of ISO 10303 makes use of the following terms defined in ISO 10303-1.

— abstract test suite;

— application;

— application activity model;

— application context;

— application interpreted model;

— application object;

— application protocol;

— application reference model;

— assembly;

— component;

— conformance class;

— conformance requirement;

— data;

— data exchange;

— implementation method;

— information;

— integrated resource;

— interpretation;

43) product_requires_product_category
44) product_requires_version
45) product_version_requires_approval
46) product_version_requires_person_organization
47) product_version_requires_security_classification
48) restrict_action_request_status
49) restrict_approval_status
50) restrict_certification_type
51) restrict_contract_type
52) restrict_date_time_role
53) restrict_document_type
54) restrict_person_organization_role
55) restrict_product_category_value
56) restrict_security_classification_level
57) security_classification_optional_date_time
58) security_classification_requires_approval
59) security_classification_requires_date_time
60) security_classification_requires_person_organization
61) start_request_requires_approval
62) start_request_requires_date_time
63) start_request_requires_person_organization
64) start_work_requires_approval
65) start_work_requires_date_time
66) subtype_mandatory_action
67) subtype_mandatory_effectivity
68) subtype_mandatory_product_context
69) subtype_mandatory_product_definition_formation
70) subtype_mandatory_product_definition_usage
71) subtype_mandatory_representation
72) subtype_mandatory_representation_context
73) subtype_mandatory_shape_representation
74) unique_version_change_order_rule
75) versioned_action_request_requires_solution
76) versioned_action_request_requires_status

5.2 AIM EXPRESS short listing

This clause specifies the EXPRESS schema that uses elements from the integrated resources and contains the types, entity specializations, rules, and functions that are specific to this part of ISO 10303. This clause also specifies modifications to the textual material for constructs that are imported from the integrated resources. The definitions and EXPRESS provided in the integrated resources for constructs used in the AIM may include select list items and subtypes which are not imported into the AIM. Requirements stated in the

integrated resources which refer to such items and subtypes apply exclusively to those items which are imported into the AIM.

EXPRESS specification:

```
*)
SCHEMA config_control_design;

  USE FROM application_context_schema   -- ISO 10303-41
      (application_context,
       application_protocol_definition,
       product_context,
       product_definition_context,
       product_concept_context);

  USE FROM product_definition_schema   -- ISO 10303-41
      (product,
       product_definition,
       product_definition_formation,
       product_definition_formation_with_specified_source,
       product_definition_relationship,
       product_category,
       product_category_relationship,
       product_related_product_category,
       product_definition_with_associated_documents);

  USE FROM product_structure_schema  -- ISO 10303-44
      (product_definition_usage,
       assembly_component_usage,
       next_assembly_usage_occurrence,
       promissory_usage_occurrence,
       quantified_assembly_component_usage,
       specified_higher_usage_occurrence,
       assembly_component_usage_substitute,
       alternate_product_relationship);

  USE FROM configuration_management_schema  -- ISO 10303-44
      (configuration_item,
       configuration_design,
       configuration_effectivity);
```

```
USE FROM product_concept_schema  -- ISO 10303-44
    (product_concept);

USE FROM product_property_definition_schema  -- ISO 10303-41
    (product_definition_shape,
     property_definition,
     shape_aspect,
     shape_aspect_relationship);

USE FROM product_property_representation_schema  -- ISO 10303-41
    (context_dependent_shape_representation,
     property_definition_representation,
     shape_representation,
     shape_representation_relationship,
     shape_definition_representation);

USE FROM representation_schema  -- ISO 10303-43
    (functionally_defined_transformation,
     item_defined_transformation,
     global_uncertainty_assigned_context,
     mapped_item,
     representation,
     representation_context,
     parametric_representation_context,
     representation_item,
     representation_map,
     representation_relationship,
     representation_relationship_with_transformation,
     using_representations);

USE FROM geometry_schema  -- ISO 10303-42
    (axis1_placement,
     axis2_placement_2d,
     axis2_placement_3d,
     b_spline_curve,
     b_spline_curve_with_knots,
     b_spline_surface,
     b_spline_surface_with_knots,
     bezier_curve,
     bezier_surface,
     boundary_curve,
     cartesian_point,
```

```
cartesian_transformation_operator_3d,
circle,
composite_curve,
composite_curve_on_surface,
composite_curve_segment,
conic,
conical_surface,
curve,
curve_bounded_surface,
curve_replica,
cylindrical_surface,
degenerate_pcurve,
degenerate_toroidal_surface,
dimension_count,
dimension_of,
direction,
ellipse,
evaluated_degenerate_pcurve,
geometric_representation_context,
geometric_representation_item,
hyperbola,
intersection_curve,
line,
offset_curve_3d,
offset_surface,
outer_boundary_curve,
parabola,
pcurve,
plane,
point,
point_on_curve,
point_on_surface,
point_replica,
polyline,
quasi_uniform_curve,
quasi_uniform_surface,
rational_b_spline_curve,
rational_b_spline_surface,
rectangular_composite_surface,
rectangular_trimmed_surface,
reparametrised_composite_curve_segment,
seam_curve,
```

```
      spherical_surface,
      surface,
      surface_curve,
      surface_of_linear_extrusion,
      surface_of_revolution,
      surface_replica,
      swept_surface,
      toroidal_surface,
      trimmed_curve,
      uniform_curve,
      uniform_surface,
      vector);

  USE FROM topology_schema  -- ISO 10303-42
     (closed_shell,
      connected_edge_set,
      connected_face_set,
      edge_curve,
      edge_loop,
      face_bound,
      face_outer_bound,
      face_surface,
      open_shell,
      oriented_closed_shell,
      oriented_face,
      path,
      poly_loop,
      topological_representation_item,
      vertex_loop,
      vertex_point,
      vertex_shell,
      wire_shell);

  USE FROM geometric_model_schema  -- ISO 10303-42
     (brep_with_voids,
      edge_based_wireframe_model,
      faceted_brep,
      geometric_curve_set,
      geometric_set,
      manifold_solid_brep,
      shell_based_surface_model,
      shell_based_wireframe_model);
```

```
USE FROM action_schema  -- ISO 10303-41
    (action,
     action_method,
     action_request_solution,
     action_request_status,
     action_status,
     action_directive,
     directed_action,
     versioned_action_request);

USE FROM certification_schema  -- ISO 10303-41
    (certification,
     certification_type);

USE FROM approval_schema  -- ISO 10303-41
    (approval_date_time,
     approval_person_organization,
     approval,
     approval_status,
     approval_relationship);

USE FROM contract_schema  -- ISO 10303-41
    (contract,
     contract_type);

USE FROM security_classification_schema   -- ISO 10303-41
    (security_classification,
     security_classification_level);

USE FROM person_organization_schema  -- ISO 10303-41
    (person_and_organization,
     organization_relationship,
     personal_address,
     organizational_address,
     organizational_project,
     person_and_organization_role);

USE FROM date_time_schema  -- ISO 10303-41
    (date_and_time,
     date,
     calendar_date,
     ordinal_date,
```

```
      week_of_year_and_day_date,
      date_time_role);

  USE FROM document_schema  -- ISO 10303-41
     (document_with_class,
      document_usage_constraint,
      document_type,
      document_relationship);

  USE FROM effectivity_schema  -- ISO 10303-41
     (effectivity,
      serial_numbered_effectivity,
      dated_effectivity,
      lot_effectivity);

  USE FROM management_resources_schema  -- ISO 10303-41
     (approval_assignment,
      certification_assignment,
      contract_assignment,
      date_and_time_assignment,
      person_and_organization_assignment,
      document_reference,
      security_classification_assignment,
      action_assignment,
      action_request_assignment);

  USE FROM measure_schema -- ISO 10303-41
     (measure_value,
      area_measure,
      count_measure,
      descriptive_measure,
      context_dependent_measure,
      parameter_value,
      plane_angle_measure,
      positive_length_measure,
      positive_plane_angle_measure,
      mass_measure,
      solid_angle_measure,
      volume_measure,
      named_unit,
      context_dependent_unit,
      conversion_based_unit,
```

```
        si_unit,
        area_unit,
        length_unit,
        mass_unit,
        plane_angle_unit,
        solid_angle_unit,
        volume_unit,
        measure_with_unit,
        area_measure_with_unit,
        length_measure_with_unit,
        mass_measure_with_unit,
        plane_angle_measure_with_unit,
        solid_angle_measure_with_unit,
        volume_measure_with_unit,
        global_unit_assigned_context);

USE FROM aic_edge_based_wireframe;   -- ISO 10303-501

USE FROM aic_shell_based_wireframe;   -- ISO 10303-502

USE FROM aic_geometrically_bounded_surface;   -- ISO 10303-507

USE FROM aic_manifold_surface;   -- ISO 10303-509

USE FROM aic_geometrically_bounded_wireframe;   -- ISO 10303-510

USE FROM aic_topologically_bounded_surface;   -- ISO 10303-511

USE FROM aic_faceted_brep;   -- ISO 10303-512

USE FROM aic_advanced_brep;   -- ISO 10303-514
(*
```

NOTE - The schemas referenced above can be found in the following parts of ISO 10303:

application_context_schema	ISO 10303-41
product_definition_schema	ISO 10303-41
product_structure_schema	ISO 10303-44
configuration_management_schema	ISO 10303-44
product_concept_schema	ISO 10303-44
product_property_definition_schema	ISO 10303-41
product_property_representation_schema	ISO 10303-41

representation_schema	ISO 10303-43
geometry_schema	ISO 10303-42
geometric_model_schema	ISO 10303-42
action_schema	ISO 10303-41
certification_schema	ISO 10303-41
approval_schema	ISO 10303-41
contract_schema	ISO 10303-41
security_classification_schema	ISO 10303-41
person_organization_schema	ISO 10303-41
date_time_schema	ISO 10303-41
document_schema	ISO 10303-41
effectivity_schema	ISO 10303-41
management_resources_schema	ISO 10303-41
measure_schema	ISO 10303-41
aic_edge_based_wireframe	ISO 10303-501
aic_shell_based_wireframe	ISO 10303-502
aic_geometrically_bounded_surface	ISO 10303-507
aic_manifold_surface	ISO 10303-509
aic_geometrically_bounded_wireframe	ISO 10303-510
aic_topologically_bounded_surface	ISO 10303-511
aic_faceted_brep	ISO 10303-512
aic_advanced_brep	ISO 10303-514

5.2.1 Fundamental concepts and assumptions

ISO 10303-203 is designed to be used in the configuration control of three dimensional product design data. The fundamental concept of this schema is that the organization controls the configuration of these types of product designs. This schema is not designed to control product designs through drawing control. It is designed to provide a vehicle to control the configuration of the three dimensional design.

5.2.1.1 Relating the shape of a product to its configuration data

The shape of products in this part of ISO 10303 is represented by the **shape_representation** entity. This entity and its subtypes define the geometric and/or topological entities which make up a particular representation type. Each mechanical part or assembly that is of interest will be given by an instance of the **product** entity. Each product will, in turn, have at least one version given by an instance of the **product_-definition_formation** entity. Each version may have one or more definitions given by the **product_-definition** entity. Each definition may have its shape represented. This is done using the AIM entities by relating the **product_definition** instance to the appropriate **shape_representation** instance. The concept of the shape of a **product_definition** is given by an instance of the entity **product_definition_shape**. That shape of the product is then connected to the **shape_representation** entity by an instance of **shape_-definition_representation**. A **shape_definition_representation** inherits attributes from its supertype

property_definition_representation that reference the **shape_representation** entity that contains the geometry and/or topology for the shape of the part and a select type called **characterized_definition**. This type allows for the representation of a **shape_aspect**, **shape_aspect_relationship** or a **characterized_-product_definition** which is also a select type. To specify that the property of interest is the shape of the **product_definition**, the **product_definition_shape** subtype of **property_definition** shall be used. The **product_definition_shape** entity is constrained to use the **characterized_product_definition** select type. The **characterized_product_definition** select type allows for the representation of a **product_definition** or a **product_definition_relationship** via a reference through its select list. This reference ensures that the **product_definition_shape** entity will be used to define the shape of a **product_definition**. If the property of interest is an aspect of the shape of the **product_definition** or the relationship between two aspects of the shape of the **product_definition**, the **property_definition** shall be used and its **definition** attribute shall reference an instance of **shape_aspect** or **shape_aspect_relationship**. In order to specify the shape of a product, the **product_definition** entity will be referenced here. The constraint **subtype_mandatory_shape_representation** (see 5.2.5.73) specifies that one of the subtypes of **shape_representation** shall be used to specify the shape of the part. The representation of a **shape_aspect** or **shape_aspect_relationship** may be given by any set of **representation_item**s.

5.2.1.2 Relating the shape of a component to the shape of its assembly

There are two methods that may be used to relate a component part's shape to the shape of the assembly part in which it is assembled. The first method consists of defining the shape for each part (component and assembly), and then relating the two shapes and providing the information that defines the orientation of the component part with respect to the assembly part through a transformation. The second method consists of defining the shape for each part (component and assembly), and then incorporating the shape of the component directly in the shape of the assembly. The first method shall be used to relate the shapes that are represented by different representation types. The second method may be used for the incorporation of a components representation into the assembly's representation if the two types are the same.

Both methods employ the **shape_representation** and **product_definition** entities. The first method also employs the **product_definition_relationship**, **shape_representation_relationship** AND **representation_relationship_with_transformation** (an instance of each entity forming a complex entity instance of the two entities with an AND relationship) and **context_dependent_shape_representation** entities. The second method employs the **mapped_item** and **representation_map** entities.

When using the first method to relate the shape of the component to the shape of the assembly, each of the **shape_representation** entities that define the shapes of the component and assembly **product_definition**s is related through references in the **shape_representation_relationship** entity. Orientation information, in this case, will be provided by the formation of a complex instance of the **shape_representation_relationship** AND **representation_relationship_with_transformation** entities. The **representation_relationship_-with_transformation** entity references a **transformation** which is a select type allowing the orientation to be defined using an **axis2_placement_3d** entity in each representation for an **item_defined_transformation** or a **cartesian_transformation_operator** entity for a **functionally_defined_transformation**. In addition,

an instance of the **context_dependent_shape_representation** must be given to explicitly relate the **shape_-representation_relationship** that defines the relationship of the two shapes to the **product_definition_-relationship** that defines the assembly-component relationship between the two **product_definition**s.

When using the second method to relate the shape of the component to the shape of the assembly, the **shape_representation** entity that defines the geometry and/or topology for the component part's shape is referenced by an instance of a **representation_map** entity that is referenced by the **mapping_source** attribute of an instance of the **mapped_item** entity. The attribute **mapped_representation** of the **representation_map** will reference the **shape_representation** subtype that defines the geometric and/or topological representation of the shape. The instance of the **mapped_item** entity is then added to the set of **items** in the **shape_representation** entity that defines the geometry and/or topology for the assembly part.

5.2.1.3 Types of shape representation

This part of ISO 10303 defines eight types of representation for shapes of parts - wireframe representations using edge based and shell based models, wireframe representations that are geometrically bounded, manifold surface models, surface models that are geometrically bounded, faceted boundary representation solid models and boundary representation solid models. Each of these types of representation is self contained meaning that one type may not contain another type. Each of the types is given by a subtype of the **shape_-representation** entity. Each subtype contains local rules that govern the types of geometric and/or topological entities that can be used in it. Every **shape_representation** must be one of the subtypes unless it is used as the representation of the shape of an assembly used in method 1 above. In that case, the **shape_-representation** instance will contain only **axis2_placement_3d** entities in its set of **items** in order to define the orientation of the components' representations in it. Since the rules in each of the subtypes will conflict with each other, any **shape_representation** that is referenced by a **representation_map** to implement method 2 above must be of the same type as the **shape_representation** that has the **mapped_item** that references that **representation_map** in its set of **items**.

5.2.1.4 Use of global rules

Many of the relationships among different entities in the integrated resource parts of ISO 10303 are specified using the most generic cardinality of zero or more between two related entities. This cardinality means that the relationship is optional or there may be one or more instances of a related entity that is related to a single instance of the relating entity. This part of ISO 10303 uses global rules to constrain that cardinality. In some cases the constraint is made to be one to one, and in some instances at least one. Examples of these rules include **contract_requires_person_organization**, **approval_requires_approval_date_time**, and **certification_requires_approval** for exactly one, and **change_request_requires_person_organization** and **product_requires_version** for one or more.

Global rules are also used to restrict the values of STRING type attributes to be only those that are applicable within the context of configuration controlled 3d design of mechanical parts and assemblies. Examples of these rules are **restrict_approval_status** and **restrict_person_organization_role**.

5.2.1.5 Assignment of units

Units are assigned to the representation of shape on a global basis. This is done by the creation of an instance of the **global_unit_assigned_context**. This entity contains an attribute that allows for a set of units to be assigned to a **representation_context**. Each of the **shape_representation** entities has a context for its representation. If units of measure are desired for a particular instance of **shape_representation**, then that instance shall have a **global_unit_assigned_context** in its **context_of_items** attribute.

5.2.2 Configuration controlled design constants

EXPRESS specification:

```
*)
CONSTANT
(*
```

5.2.2.1 dummy_gri

A **dummy_gri** identifies a **geometric_representation_item** which has a null **name** value for use in constructor functions.

EXPRESS specification:

```
*)
  dummy_gri : geometric_representation_item := representation_item('') ||
              geometric_representation_item();
(*
```

5.2.2.2 dummy_tri

A **dummy_tri** identifies a **topological_representation_item** which has a null **name** value for use in constructor functions.

EXPRESS specification:

```
*)
  dummy_tri : topological_representation_item := representation_item('') ||
              topological_representation_item();
END_CONSTANT;
(*
```

5.2.3 Configuration controlled design types

5.2.3.1 work_item

A **work_item** identifies the **product_definition_formation** that is the result of initial design activity or a modification to a design.

EXPRESS specification:

```
*)
TYPE work_item = SELECT (product_definition_formation);
END_TYPE;
(*
```

5.2.3.2 change_request_item

A **change_request_item** is the **product_definition_formation** of the part that is to be affected by the **change_request**.

EXPRESS specification:

```
*)
TYPE change_request_item = SELECT (product_definition_formation);
END_TYPE;
(*
```

5.2.3.3 start_request_item

A **start_request_item** is the **product_definition_formation** of the part that is to be created by the **start_-request**.

EXPRESS specification:

```
*)
TYPE start_request_item = SELECT (product_definition_formation);
END_TYPE;
(*
```

5.2.3.4 certified_item

A **certified_item** applies a **certification** to a part that is supplied by an external organization. The **certification** stipulates that the external organization is qualified to produce the part.

EXPRESS specification:

```
*)
TYPE certified_item = SELECT (supplied_part_relationship);
END_TYPE;
(*
```

5.2.3.5 approved_item

An **approved_item** assigns an **approval** to a **product_definition_formation**, **product_definition**, **planned_effectivity**, **configuration_item**, **security_classification**, **change_request**, **change**, **start_-request**, **start_work**, **certification**, or **contract** to indicate the approval status of the selected aspect of the design.

EXPRESS specification:

```
*)
TYPE approved_item = SELECT
  (product_definition_formation,
   product_definition,
   configuration_effectivity,
   configuration_item,
   security_classification,
   change_request,
   change,
   start_request,
   start_work,
   certification,
   contract);
END_TYPE;
(*
```

5.2.3.6 contracted_item

A **contracted_item** associates a particular **product_definition_formation** with a **contract**.

EXPRESS specification:

```
*)
TYPE contracted_item = SELECT (product_definition_formation);
END_TYPE;
(*
```

5.2.3.7 classified_item

A **classified_item** applies **security_classification** to a particular **product_definition_formation** or a relationship between two **product_definition**s for a particular usage.

EXPRESS specification:

```
*)
TYPE classified_item = SELECT
  (product_definition_formation,
   assembly_component_usage);
END_TYPE;
(*
```

5.2.3.8 person_organization_item

A **person_organization_item** assigns a **person_and_organization** to a **change_request**, **start_request**, **approval**, **configuration_item**, **product**, **product_definition_formation**, **product_definition**, **contract**, or **security_classification**. The role for the **person_and_organization** is controlled by the **cc_design_-person_and_organization_correlation** function given in 5.2.6.2.

EXPRESS specification:

```
*)
TYPE person_organization_item = SELECT
  (change,
   start_work,
   change_request,
   start_request,
   configuration_item,
   product,
   product_definition_formation,
   product_definition,
   contract,
   security_classification);
```

```
END_TYPE;
(*
```

5.2.3.9 date_time_item

A **date_time_item** assigns a **date_and_time** to a **product_definition**, **change_request**, **start_request**, **change**, **start_work**, **approval_person_organization**, **contract**, **security_classification**, or **certification**. The role for the **date_and_time** is controlled by the **cc_design_date_time_correlation** function given in 5.2.6.3.

EXPRESS specification:

```
*)
TYPE date_time_item = SELECT
  (product_definition,
   change_request,
   start_request,
   change,
   start_work,
   approval_person_organization,
   contract,
   security_classification,
   certification);
END_TYPE;
(*
```

5.2.3.10 specified_item

A **specified_item** assigns a **specification** to either a **product_definition** or a **shape_aspect**.

EXPRESS specification:

```
*)
TYPE specified_item = SELECT
  (product_definition,
   shape_aspect);
END_TYPE;
(*
```

5.2.4 Configuration controlled design entities

5.2.4.1 Configuration controlled design entity definitions

5.2.4.1.1 mechanical_context

A **mechanical_context** is a **product_context** that is applicable to those products that are mechanical.

NOTE - The use of this entity defines a viewpoint for the context of a product. It is not intended to be a classification or categorization of a type of product. The definition of the **product_context** to be mechanical using this entity is specifying the way that the product acts within the data exchange.

EXAMPLE 21 - A printed circuit assembly is defined in a mechanical context when its physical properties such as shape are of interest, and are defined so that its fit within the assembly in which it used may be specified. Other properties of the printed circuit assembly such as connectivity requirements and functional requirements would not be specified in a mechanical context.

EXPRESS specification:

```
*)
ENTITY mechanical_context
  SUBTYPE OF (product_context);
WHERE
  WR1: SELF.discipline_type = 'mechanical';
END_ENTITY;
(*
```

Formal propositions:

WR1: the **discipline_type** of the **mechanical_context** entity shall contain the value 'mechanical'.

5.2.4.1.2 design_context

A **design_context** is a **product_definition_context** that defines a life cycle stage of design as the frame of reference for **product_definition** entities.

NOTE - A further refinement within the context of design may be made using the **application_context_-element** entity.

EXPRESS specification:

```
*)
ENTITY design_context
  SUBTYPE OF (product_definition_context);
WHERE
  WR1: SELF.life_cycle_stage = 'design';
END_ENTITY;
(*
```

Formal propositions:

WR1: the **life_cycle_stage** of the **design_context** entity shall contain the value 'design'.

5.2.4.1.3 design_make_from_relationship

A **design_make_from_relationship** specifies that one **product**'s design has been derived from another **product**'s design. The use of the **design_make_from_relationship** also implies that the physical product resulting from the manufacture of the **relating_product_definition** is to be used as the basis for the manufacture of the **related_product_definition**.

EXPRESS specification:

```
*)
ENTITY design_make_from_relationship
  SUBTYPE OF (product_definition_relationship);
END_ENTITY;
(*
```

Attribute definitions:

SELF\product_definition_relationship.relating_product_definition: the **product_definition** that is the source for the make from relationship.

SELF\product_definition_relationship.related_product_definition: the **product_definition** entity that is created from the source **product_definition**.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **design_make_from_relationship** entity:

— no_shape_for_make_from (See 5.2.5.76).

5.2.4.1.4 supplied_part_relationship

A **supplied_part_relationship** relates the identifications of two **product**s and specifies that one is the identification used by the internal design organization and the other is the identification used by a supplier.

EXPRESS specification:

```
*)
ENTITY supplied_part_relationship
  SUBTYPE OF (product_definition_relationship);
END_ENTITY;
(*
```

Attribute definitions:

SELF\product_definition_relationship.relating_product_definition: the **product_definition** that is the design organization's definition and identification of the **product**.

SELF\product_definition_relationship.related_product_definition: the **product_definition** that is definition and identification of the product for the organization which supplies the part or design to the design organization.

Informal propositions:

IP1: The shapes for design organization's definition and the supplying organization's definition shall be equal with respect to form, fit and function.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **supplied_part_relationship** entity:

— no_shape_for_supplied_part (See 5.2.5.77).

5.2.4.1.5 change_request

A **change_request** is a formal notification of a desire for a modification to a particular piece of product data.

EXPRESS specification:

```
*)
ENTITY change_request
  SUBTYPE OF (action_request_assignment);
  items : SET [1:?] OF change_request_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **change_request_item**s which identify versions of particular **product**s to be changed.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **change_request** entity:

— change_request_requires_approval (See 5.2.5.6);

— change_request_requires_person_organization (See 5.2.5.7);

— change_request_requires_date_time (See 5.2.5.8).

5.2.4.1.6 start_request

A **start_request** is a formal notification of a desire to begin a new design for a **product** or group of **product**s.

EXPRESS specification:

```
*)
ENTITY start_request
  SUBTYPE OF (action_request_assignment);
  items : SET [1:?] OF start_request_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **start_request_item**s which identify the versions of particular **product**s.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **start_request** entity:

— start_request_requires_approval (See 5.2.5.11);

— start_request_requires_person_organization (See 5.2.5.12);

— start_request_requires_date_time (See 5.2.5.13).

5.2.4.1.7 change

A **change** identifies the **change_request**s that have been incorporated into the design and, as a result, have established a new version of the **product**.

EXPRESS specification:

```
*)
ENTITY change
  SUBTYPE OF (action_assignment);
  items : SET [1:?] OF work_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **work_item**s which identify versions of particular **product**s which were created as a result of the incorporation of the **change**.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **change** entity:

— change_requires_approval (See 5.2.5.9);

— change_requires_date_time (See 5.2.5.10);

— unique_version_change_order_rule (See 5.2.5.19).

5.2.4.1.8 start_work

A **start work** identifies the **start_request**s that have been fulfilled and, as a result, have established a new

version of a **product**.

EXPRESS specification:

```
*)
ENTITY start_work
  SUBTYPE OF (action_assignment);
  items : SET [1:?] OF work_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **work_item**s which identify versions of **product**s that are to be created as a result of the start of the **work**.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **start_work** entity:

— start_work_requires_approval (See 5.2.5.14);

— start_work_requires_date_time (See 5.2.5.15).

5.2.4.1.9 cc_design_certification

A **cc_design_certification** assigns a **certification** to a **product** that is supplied by an external organization.

EXPRESS specification:

```
*)
ENTITY cc_design_certification
  SUBTYPE OF (certification_assignment);
  items : SET [1:?] OF certified_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **certified_item**s which identify **supplied_part_relationship** entities to which the **certification** is being assigned.

5.2.4.1.10 cc_design_approval

A **cc_design_approval** relates an **approval** to a particular piece of product data.

EXPRESS specification:

```
*)
ENTITY cc_design_approval
  SUBTYPE OF (approval_assignment);
  items : SET [1:?] OF approved_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **approved_item**s which identify the particular **product_definition_formation**, **product_-definition**, **planned_effectivity**, **configuration_item**, **security_classification**, **change_request**, **change**, **start_request, start_work**, **certification**, or **contract** to which the **approval** applies.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **cc_design_approval** entity:

— change_request_requires_approval (See 5.2.5.6);

— change_requires_approval (See 5.2.5.9);

— start_request_requires_approval (See 5.2.5.11);

— start_work_requires_approval (See 5.2.5.14);

— product_version_requires_approval (See 5.2.5.22);

— product_definition_requires_approval (See 5.2.5.26);

— certification_requires_approval (See 5.2.5.28);

— contract_requires_approval (See 5.2.5.35);

— security_classification_requires_approval (See 5.2.5.38);

— effectivity_requires_approval (See 5.2.5.66);

— configuration_item_requires_approval (See 5.2.5.67).

5.2.4.1.11 cc_design_contract

A **cc_design_contract** relates a **contract** to a **product_definition_formation**.

EXPRESS specification:

```
*)
ENTITY cc_design_contract
  SUBTYPE OF (contract_assignment);
  items : SET [1:?] OF contracted_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **contracted_item**s which identify the versions of particular **product**s to which the **contract** is assigned.

5.2.4.1.12 cc_design_security_classification

A **cc_design_security_classification** relates a **security_classification** to a particular piece of product data.

EXPRESS specification:

```
*)
ENTITY cc_design_security_classification
  SUBTYPE OF (security_classification_assignment);
  items : SET [1:?] OF classified_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **classified_item**s which identify the versions or definition relationships of particular **product**s to which the **security_classification** is assigned.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **cc_design_security_classification** entity:

— product_version_requires_security_classification (See 5.2.5.24);

— pdu_requires_security_classification (See 5.2.5.70).

5.2.4.1.13 cc_design_person_and_organization_assignment

A **cc_design_person_and_organization_assignment** relates a **person_and_organization** to a particular piece of product data.

EXPRESS specification:

```
*)
ENTITY cc_design_person_and_organization_assignment
  SUBTYPE OF (person_and_organization_assignment);
  items : SET [1:?] OF person_organization_item;
WHERE
  WR1: cc_design_person_and_organization_correlation (SELF);
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **person_organization_item**s which identify the **change_request**, **start_request**, **configuration_item**, **product**, **product_definition_formation**, **contract**, or **security_classification** to which the **person_and_organization** is related.

Formal propositions:

WR1: The **cc_design_person_and_organization_correlation** function that correlates roles of persons and organizations to elements of product data shall be satisfied. See 5.2.6.2 for the complete definition of the **cc_design_person_and_organization_correlation** function.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **cc_design_person_and_-organization_assignment** entity:

— change_request_requires_person_organization (See 5.2.5.7);

— start_request_requires_person_organization (See 5.2.5.12);

— product_requires_person_organization (See 5.2.5.21);

— product_version_requires_person_organization (See 5.2.5.23);

— product_definition_requires_person_organization (See 5.2.5.25);

— contract_requires_person_organization (See 5.2.5.36);

— security_classification_requires_person_organization (See 5.2.5.39);

— configuration_item_requires_person_organization (See 5.2.5.64).

5.2.4.1.14 cc_design_date_and_time_assignment

A **cc_design_date_and_time_assignment** relates a **date_and_time_assignment** to a particular piece of product data.

EXPRESS specification:

```
*)
ENTITY cc_design_date_and_time_assignment
  SUBTYPE OF (date_and_time_assignment);
  items : SET [1:?] OF date_time_item;
WHERE
  WR1: cc_design_date_time_correlation (SELF);
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **date_time_item**s which identify the particular **product_definition**, **change_request**, **start_-request**, **change**, **start_work**, **contract**, **security_classification**, **change**, **certification** or **approval_-person_and_organization** to which the date and time is assigned.

Formal propositions:

WR1: The **cc_design_date_time_correlation** function which correlates roles of dates and times to elements of product data shall be satisfied. See 5.2.6.3 the complete definition of the **cc_design_date_time_-correlation** function.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **cc_design_date_and_time_-assignment** entity:

— change_request_requires_date_time (See 5.2.5.8);

— change_requires_date_time (See 5.2.5.10);

— start_request_requires_date_time (See 5.2.5.13);

— start_work_requires_date_time (See 5.2.5.15);

— product_definition_requires_date_time (See 5.2.5.27);

— certification_requires_date_time (See 5.2.5.30);

— security_classification_requires_date_time (See 5.2.5.40);

— security_classification_optional_date_time (See 5.2.5.41).

5.2.4.1.15 cc_design_specification_reference

A **cc_design_specification_reference** relates a reference to a specification to a complete definition of a product or to some aspect of its shape.

EXPRESS specification:

```
*)
ENTITY cc_design_specification_reference
  SUBTYPE OF (document_reference);
  items : SET [1:?] OF specified_item;
END_ENTITY;
(*
```

Attribute definitions:

items: a set of **product_definition**s and / or **shape_aspect**s to which the **specification_reference** applies.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **cc_design_specification_-reference** entity:

— document_to_product_definition (See 5.2.5.46).

5.2.4.2 Configuration controlled design imported entity modifications

5.2.4.2.1 application_context

The base definition of the **application_context** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **application_context** entity:

— application_context_requires_ap_definition (See 5.2.5.1).

5.2.4.2.2 application_protocol_definition

The base definition of the **application_protocol_definition** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **application_protocol_definition** entity:

— application_context_requires_ap_definition (See 5.2.5.1).

5.2.4.2.3 product_context

The base definition of the **product_context** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **product_context** entity:

— subtype_mandatory_product_context (See 5.2.5.2).

5.2.4.2.4 product_definition_context

The base definition of the **product_definition_context** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

The definition of **product_definition_context** is modified as follows:

The **application_context_element** attribute **name** shall be used to modify the **product_definition_context** in order to specify the exact context within design to which the **product_definition** is applicable.

5.2.4.2.5 product_category

The base definition of the **product_category** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

The definition of **product_category** is modified as follows:

In this part of ISO 10303, **product_category** trees are utilized to categorize parts. An intermediate node in the tree may used to segregate standard parts from non-standard parts. In this case, the **name** attribute for the **product_category** shall be "standard_part".

5.2.4.2.6 product_related_product_category

The base definition of the **product_related_product_category** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **product_related_product_-category** entity:

— restrict_product_category_value (See 5.2.5.4);

— product_requires_product_category (See 5.2.5.5).

5.2.4.2.7 product

The base definition of the **product** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **product** entity:

— product_requires_product_category (See 5.2.5.5);

— product_requires_version (See 5.2.5.20);

— product_requires_person_organization (See 5.2.5.21).

5.2.4.2.8 product_definition_formation

The base definition of the **product_definition_formation** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **product_definition_formation** entity:

— product_requires_version (See 5.2.5.20);

— product_version_requires_approval (See 5.2.5.22);

— product_version_requires_person_organization (See 5.2.5.23);

— product_version_requires_security_classification (See 5.2.5.24);

— subtype_mandatory_product_definition_formation (See 5.2.5.50).

5.2.4.2.9 product_definition

The base definition of the **product_definition** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **product_definition** entity:

— design_context_for_property (See 5.2.5.3);

— product_definition_requires_person_organization (See 5.2.5.25);

— product_definition_requires_approval (See 5.2.5.26);

— product_definition_requires_date_time (See 5.2.5.27);

— document_to_product_definition (See 5.2.5.46).

5.2.4.2.10 action_request_status

The base definition of the **action_request_status** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **action_request_status** entity:

— restrict_action_request_status (See 5.2.5.16);

— versioned_action_request_requires_status (See 5.2.5.17).

5.2.4.2.11 versioned_action_request

The base definition of the **versioned_action_request** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **versioned_action_request** entity:

— versioned_action_request_requires_status (See 5.2.5.17);

— versioned_action_request_requires_solution (See 5.2.5.18).

5.2.4.2.12 action_request_solution

The base definition of the **action_request_solution** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **action_request_solution** entity:

— versioned_action_request_requires_solution (See 5.2.5.18).

5.2.4.2.13 certification_type

The base definition of the **certification_type** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **certification_type** entity:

— restrict_certification_type (See 5.2.5.29);

— dependent_instantiable_certification_type (See 5.2.5.62).

5.2.4.2.14 certification

The base definition of the **certification** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **certification** entity:

— certification_requires_approval (See 5.2.5.28);

— certification_requires_date_time (See 5.2.5.30).

5.2.4.2.15 approval_status

The base definition of the **approval_status** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **approval_status** entity:

— restrict_approval_status (See 5.2.5.34);

— dependent_instantiable_approval_status (See 5.2.5.59).

5.2.4.2.16 approval

The base definition of the **approval** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **approval** entity:

— approvals_are_assigned (See 5.2.5.31);

— approval_requires_approval_person_organization (See 5.2.5.32);

— approval_requires_approval_date_time (See 5.2.5.33).

5.2.4.2.17 approval_assignment

The base definition of the **approval_assignment** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **approval_assignment** entity:

— approvals_are_assigned (See 5.2.5.31).

5.2.4.2.18 approval_person_organization

The base definition of the **approval_person_organization** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rules defined in this part of ISO 10303 apply to the **approval_person_organization** entity:

— approval_person_organization_constraints (See 5.2.5.79);
— approval_requires_approval_person_organization (See 5.2.5.32).

5.2.4.2.19 approval_date_time

The base definition of the **approval_date_time** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rules defined in this part of ISO 10303 apply to the **approval_date_time** entity:

— approval_date_time_constraints (See 5.2.5.78);
— approval_requires_approval_date_time (See 5.2.5.33).

5.2.4.2.20 contract_type

The base definition of the **contract_type** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **contract_type** entity:

— restrict_contract_type (See 5.2.5.37);

— dependent_instantiable_contract_type (See 5.2.5.61).

5.2.4.2.21 contract

The base definition of the **contract** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **contract** entity:

— contract_requires_approval (See 5.2.5.35);

— contract_requires_person_organization (See 5.2.5.36).

5.2.4.2.22 security_classification_level

The base definition of the **security_classification_level** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **security_classification_level** entity:

— restrict_security_classification_level (See 5.2.5.42);

— dependent_instantiable_security_classification_level (See 5.2.5.58).

5.2.4.2.23 security_classification

The base definition of the **security_classification** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **security_classification** entity:

— security_classification_requires_approval (See 5.2.5.38);

— security_classification_requires_person_organization (See 5.2.5.39);

— security_classification_requires_date_time (See 5.2.5.40);

— security_classification_optional_date_time (See 5.2.5.41).

5.2.4.2.24 person_and_organization_role

The base definition of the **person_and_organization_role** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **person_and_organization_role** entity:

— restrict_person_organization_role (See 5.2.5.43);

— dependent_instantiable_person_and_organization_role (See 5.2.5.56).

5.2.4.2.25 person

The base definition of the **person** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Attribute definitions:

The attribute definitions are modified as follows:

id: a means by which the **person** may be identified.

NOTE - The uniqueness of this attribute must be maintained and guaranteed over the entire domain of the data. With a single organization, country or team, this constraint may be easy to satisfy. In situations where global or universal uniqueness is required, the reader is referred to ISO/IEC 8824-1 for guidance on registration of strings for international application uniqueness.

5.2.4.2.26 date_time_role

The base definition of the **date_time_role** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **date_time_role** entity:

— restrict_date_time_role (See 5.2.5.44);

— dependent_instantiable_date_time_role (See 5.2.5.55).

5.2.4.2.27 document_type

The base definition of the **document_type** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **document_type** entity:

— restrict_document_type (See 5.2.5.45);

— dependent_instantiable_document_type (See 5.2.5.60).

5.2.4.2.28 measure_with_unit

The base definition of the **measure_with_unit** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **measure_with_unit** entity:

— as_required_quantity (See 5.2.5.47).

5.2.4.2.29 global_unit_assigned_context

The base definition of the **global_unit_assigned_context** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **global_unit_assigned_context** entity:

— global_unit_assignment (See 5.2.5.48).

5.2.4.2.30 action

The base definition of the **action** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **action** entity:

— subtype_mandatory_action (See 5.2.5.49).

5.2.4.2.31 action_directive

The base definition of the **action_directive** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **action_directive** entity:

— dependent_instantiable_action_directive (See 5.2.5.57).

5.2.4.2.32 date

The base definition of the **date** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **date** entity:

— dependent_instantiable_date (See 5.2.5.51).

5.2.4.2.33 representation

The base definition of the **representation** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **representation** entity:

— subtype_mandatory_representation (See 5.2.5.74).

5.2.4.2.34 representation_context

The base definition of the **representation_context** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **representation_context** entity:

— subtype_mandatory_representation_context (See 5.2.5.75).

5.2.4.2.35 shape_representation

The base definition of the **shape_representation** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **shape_representation** entity:

— dependent_instantiable_shape_representation (See 5.2.5.52);

— subtype_mandatory_shape_representation (See 5.2.5.73).

5.2.4.2.36 context_dependent_shape_representation

The base definition of the **context_dependent_shape_representation** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

The definition of **context_dependent_shape_representation** is modified as follows:

In this part of ISO 10303, **context_dependent_shape_representation** shall be used to relate the **next_-assembly_usage_occurrence** defining the usage of a component in an assembly to the **shape_-representation_relationship** which defines the position and orientation of the component's shape in the shape definition of the assembly.

NOTE - **Context_dependent_shape_representation** may be used to relate shapes of components and assemblies regardless of what type of **shape_representation** is used for each. The **mapped_item** construct may also be used to relate the shapes of components and assemblies, but the use of **mapped_item**, in this part of ISO 10303, requires that the **shape_representation** types be the same.

5.2.4.2.37 named_unit

The base definition of the **named_unit** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **named_unit** entity:

— dependent_instantiable_named_unit (See 5.2.5.53).

5.2.4.2.38 representation_item

The base definition of the **representation_item** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **representation_item** entity:

— dependent_instantiable_representation_item (See 5.2.5.54).

5.2.4.2.39 product_definition_usage

The base definition of the **product_definition_usage** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **product_definition_usage** entity:

— subtype_mandatory_product_definition_usage (See 5.2.5.69).

5.2.4.2.40 geometric_representation_item

The base definition of the **geometric_representation_item** entity is given in ISO 10303-42. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **geometric_representation_item** entity:

— geometric_representation_item_3d (See 5.2.5.71).

5.2.4.2.41 parametric_representation_context

The base definition of the **parametric_representation_context** entity is given in ISO 10303-43. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **parametric_representation_-context** entity:

— dependent_instantiable_parametric_representation_context (See 5.2.5.72).

5.2.4.2.42 product_concept

The base definition of the **product_concept** entity is given in ISO 10303-44. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **product_concept** entity:

— product_concept_requires_configuration_item (See 5.2.5.63).

5.2.4.2.43 configuration_item

The base definition of the **configuration_item** entity is given in ISO 10303-44. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **configuration_item** entity:

— product_concept_requires_configuration_item (See 5.2.5.63);

— configuration_item_requires_person_organization (See 5.2.5.64);

— configuration_item_requires_approval (See 5.2.5.67);

5.2.4.2.44 effectivity

The base definition of the **effectivity** entity is given in ISO 10303-41. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **effectivity** entity:

— subtype_mandatory_effectivity (See 5.2.5.65);

— effectivity_requires_approval (See 5.2.5.66).

5.2.4.2.45 next_assembly_usage_occurrence

The base definition of the **next_assembly_usage_occurrence** entity is given in ISO 10303-44. The following modifications apply to this part of ISO 10303.

Associated global rule:

The following global rule defined in this part of ISO 10303 applies to the **next_assembly_usage_-occurrence** entity:

— coordinated_assembly_and_shape (See 5.2.5.68).

5.2.4.2.46 assembly_component_usage

The base definition of the **assembly_componet_usage** entity is given in ISO 10303-44. The following modifications apply to this part of ISO 10303.

Associated global rules:

The following global rules defined in this part of ISO 10303 apply to the **assembly_component_usage** entity:

— acu_requires_security_classification (See 5.2.5.70);

5.2.5 Configuration controlled design rules

5.2.5.1 application_context_requires_ap_definition

The **application_context_requires_ap_definition** rule specifies that each instance of **application_context** shall be referenced by exactly one **application_protocol_definition** that specifies this part of ISO 10303.

EXPRESS specification:

```
*)
RULE application_context_requires_ap_definition FOR
  (application_context, application_protocol_definition);
WHERE
  WR1: SIZEOF (QUERY (ac <* application_context |
       NOT (SIZEOF (QUERY (apd <* application_protocol_definition |
       (ac :=: apd.application)
       AND
       (apd.application_interpreted_model_schema_name =
       'config_control_design'))) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

application_context: identifies the set of all instances of **application_context** entities.

application_protocol_definition: identifies the set of all instances of **application_protocol_definition** entities.

Formal propositions:

WR1: For each instance of **application_context**, there shall be exactly one instance of **application_-protocol_definition** that references the instance of **application_context** as its **application** with a value of 'config_control_design' as its **application_interpreted_model_schema_name**.

5.2.5.2 subtype_mandatory_product_context

The subtype_mandatory_product_context rule specifies the permitted usage of the product context. The **mechanical_context** entity shall be used to define the **discipline_type** of a **product_context**. This usage ensures that the context of the product data communicated represents a mechanical viewpoint of the data.

EXPRESS specification:

```
*)
RULE subtype_mandatory_product_context FOR (product_context);
WHERE
  WR1: SIZEOF ( QUERY (pc <* product_context |
       NOT ('CONFIG_CONTROL_DESIGN.MECHANICAL_CONTEXT' IN TYPEOF(pc))))
       = 0;
END_RULE;
(*
```

Argument definitions:

product_context: identifies the set of all instances of **product_context** entities.

Formal propositions:

WR1: Each instance of **product_context** shall be a **mechanical_context** entity.

5.2.5.3 design_context_for_property

The **design_context_for_property** rule specifies that the only life cycle stage for which properties may be defined in this part of ISO 10303 is that of design. Product structures may be defined using any life cycle stage for the context of a **product_definition**.

EXPRESS specification:

```
*)
RULE design_context_for_property FOR (product_definition);
WHERE
  WR1: SIZEOF (QUERY (pd <* product_definition |
       (SIZEOF (USEDIN (pd, 'CONFIG_CONTROL_DESIGN.' +
       'PROPERTY_DEFINITION.DEFINITION') +
       QUERY (pdr <* USEDIN (pd, 'CONFIG_CONTROL_DESIGN.' +
       'PRODUCT_DEFINITION_RELATIONSHIP.RELATED_PRODUCT_DEFINITION') |
       SIZEOF (USEDIN (pdr, 'CONFIG_CONTROL_DESIGN.PROPERTY_DEFINITION.' +
       'DEFINITION')) >= 1)) >= 1) AND
       (NOT ('CONFIG_CONTROL_DESIGN.DESIGN_CONTEXT' IN
       TYPEOF (pd.frame_of_reference))))) = 0;
END_RULE;
(*
```

Argument definition:

product_definition: identifies the set of all instances of **product_definition.**

Formal proposition:

WR1: For each instance of **product_definition**, if that instance is referenced as the **definition** attribute of a **property_definition**, or if that instance is the **related_product_definition** in a **product_definition_-relationship** instance that is referenced by the **definition** attribute of a **property_definition**, the **frame_of_-reference** attribute for that **product_definition** instance must reference a **design_context**.

5.2.5.4 restrict_product_category_value

The **restrict_product_category_value** rule specifies the set of values that a **product_category** which is related to a **product** may contain.

EXPRESS specification:

```
*)
RULE restrict_product_category_value FOR
  (product_related_product_category);
WHERE
   WR1: SIZEOF (QUERY (prpc <*
       product_related_product_category |
       NOT( prpc.name IN ['assembly', 'detail',
       'customer_furnished_equipment', 'inseparable_assembly', 'cast',
       'coined', 'drawn', 'extruded', 'forged', 'formed', 'machined',
       'molded', 'rolled', 'sheared']))) = 0;
END_RULE;
(*
```

Argument definitions:

product_related_product_category: identifies the set of all instances of **product_related_product_-category** entities.

Formal propositions:

WR1: The **name** attribute of a **product_related_product_category** shall be either "assembly", "detail", "inseparable_assembly", "customer_furnished_equipment", "cast", "coined", "drawn", "extruded", "forged", "formed", "machined", "molded", "rolled" or "sheared".

Attribute value restriction definitions:

assembly: identifies a part that consists of a collection of other parts which are put together to satisfy a particular function.

detail: identifies a part that exists at the lowest level of the bill of materials structure.

customer_furnished_equipment: identifies a part that has been furnished to the design agency by the customer.

inseparable_assembly: identifies a part that after being put together cannot be dis-assembled without causing physical harm to at least one of the components of the assembly.

cast: identifies a class of parts that are intended to be produced by a manufacturing process in which a liquid form of material is poured into a mold and hardened.

coined: identifies a class of parts that are intended to be produced by a manufacturing process in which a liquid form of material is forced into well confined dies by high pressures which forces the material to fill the shape of the die.

drawn: identifies a class of parts that are intended to be produced by a manufacturing process in which metal is pulled in order to remove material from a raw stock and form an indenture in the material of a defined shape.

extruded: identifies a class of parts that are intended to be produced by a manufacturing process in which material is squeezed by a compressing operation that changes the shape of the raw stock.

forged: identifies a class of parts that are intended to be produced by a manufacturing process in which metal materials are worked into useful shapes under the influence of external force at a high temperature.
formed: identifies a class of parts that are intended to be produced by a manufacturing process in which metal materials are subjected to stresses which change their shape.

machined: identifies a class of parts that are intended to be produced by a manufacturing process which employs a piece of automated equipment.

molded: identifies a class of parts that are intended to be produced by a manufacturing process in which material is heated past its melting point and poured into a mold to form its shape.

rolled: identifies a class of parts that are intended to be produced by a manufacturing process in which material is passed between two rolls in order to reduce it to the thickness equal to the space between the two rolls.

sheared: identifies a class of parts that are intended to be produced by a manufacturing process in which material is cut to alter its shape.

5.2.5.5 product_requires_product_category

The **product_requires_product_category** rule specifies the requirement that each **product** shall be referenced by exactly one **product_related_product_category** which defines whether the product is an "assembly", "inseparable_assembly", "detail" or "customer_furnished_equipment".

NOTE - The **product_requires_product_category** rule constrains the relationship between **product_-related_product_category** and **product**. The rule specifies that a **product** must be referenced by exactly one **product_related_product_category** whose value is either "assembly", "inseparable_assembly", "detail" or "customer_furnished_equipment". The rule does not preclude the inclusion of other **product_related_-product_category** entity instances with values other than those stated above for the **name** attribute of the **product_related_product_category**. Valid values for the **name** attribute of the **product_related_-product_category** entity are defined in the rule **restrict_product_category_value**.

EXPRESS specification:

```
*)
RULE product_requires_product_category FOR
  (product,product_related_product_category);
WHERE
  WR1: SIZEOF (QUERY (prod <* product |
       NOT (SIZEOF (QUERY (prpc <* product_related_product_category |
       (prod IN prpc.products) AND
       (prpc.name IN ['assembly', 'inseparable_assembly', 'detail',
       'customer_furnished_equipment']))) = 1))) = 0;
END_RULE;
(*
```

Argument definitions:

product: identifies the set of all instances of **product** entities.

product_related_product_category: identifies the set of all instances of **product_related_product_-category** entities.

Formal propositions:

WR1: For each instance of **product**, there shall be exactly one instance of **product_related_product_-category** that contains a **name** attribute that has a value of either "assembly", "inseparable_assembly", "detail" or "customer_furnished_equipment".

5.2.5.6 change_request_requires_approval

The **change_request_requires_approval** rule specifies that each instance of a **change_request** shall be referenced by exactly one **cc_design_approval**. This rule enforces the requirement for each change request to have an approval.

EXPRESS specification:

```
*)
RULE change_request_requires_approval FOR
  (change_request, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (cr <* change_request |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       cr IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

change_request: identifies the set of all instances of **change_request** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **change_request**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **change_request** in its set of **items**.

5.2.5.7 change_request_requires_person_organization

The **change_request_requires_person_organization** rule specifies that every **change_request** shall be referenced by at least one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **change_request** to have a recipient who receives the request. The meaning of recipient is found in the **person_and_organization_assignment** entity's **role** attribute.

NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**.

EXPRESS specification:

```
*)
RULE change_request_requires_person_organization FOR
  (change_request,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (cr <* change_request |
       NOT (SIZEOF (QUERY (ccpoa <*
       cc_design_person_and_organization_assignment |
       cr IN ccpoa.items )) >= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

change_request: identifies the set of all instances of **change_request** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **change_request**, there shall be at least one instance of **cc_design_person_-and_organization_assignment** that contains the instance of **change_request** in its set of **items**.

5.2.5.8 change_request_requires_date_time

The **change_request_requires_date_time** rule specifies that each instance of a **change_request** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **change_request** to have a request date indicating the date on which the **change_request** was issued.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**.

EXPRESS specification:

```
*)
RULE change_request_requires_date_time FOR
  (change_request, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (cr <* change_request |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       cr IN ccdta.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

change_request: identifies the set of all instances of **change_request** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **change_request**, there shall be exactly one instance of **cc_design_date_and_-time_assignment** that contains the instance of **change_request** in its set of **items**.

5.2.5.9 change_requires_approval

The **change_requires_approval** rule specifies that each instance of **change** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every change to have an approval.

EXPRESS specification:

```
*)
RULE change_requires_approval FOR
  (change, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (chg <* change |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       chg IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

change: identifies the set of all instances of **change** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **change**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **change** in its set of **items**.

5.2.5.10 change_requires_date_time

The **change_requires_date_time** rule specifies that each instance of a **change** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **change** to have a change date indicating the date on which the **change** is incorporated into the design.

> NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**.

EXPRESS specification:

```
*)
RULE change_requires_date_time FOR
  (change, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (chg <* change |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       (chg IN ccdta.items)
       AND (ccdta.role.name = 'start_date'))) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

change: identifies the set of all instances of **change** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **change**, there shall be exactly one instance of **cc_design_date_and_time_-assignment** that contains the instance of **change** in its set of **items** where the role name is "start_date".

5.2.5.11 start_request_requires_approval

The **start_request_requires_approval** rule specifies that each instance of **start_request** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every **start_request** to have an approval which authorizes the initiation of a design.

EXPRESS specification:

```
*)
RULE start_request_requires_approval FOR
  (start_request, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (sr <* start_request |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       sr IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

start_request: identifies the set of all instances of **start_request** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **start_request**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **start_request** in its set of **items**.

5.2.5.12 start_request_requires_person_organization

The **start_request_requires_person_organization** rule specifies that every **start_request** shall be referenced by at least one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **start_request** to have a recipient who receives the request. The meaning of recipient is found in the **person_and_organization_assignment** entity's **role** attribute.

NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**.

EXPRESS specification:

```
*)
RULE start_request_requires_person_organization FOR (start_request,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (sr <* start_request |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       sr IN ccdpoa.items )) >= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

start_request: identifies the set of all instances of **start_request** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **start_request**, there shall be at least one instance of **cc_design_person_and_-organization_assignment** that contains the instance of **start_request** in its set of **items**.

5.2.5.13 start_request_requires_date_time

The **start_request_requires_date_time** rule specifies that each instance of a **start_request** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **start_request** to have a request date indicating the date on which the **start_request** was issued.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**.

EXPRESS specification:

```
*)
RULE start_request_requires_date_time FOR
  (start_request, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (sr <* start_request |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       sr IN ccdta.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

start_request: identifies the set of all instances of **start_request** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **start_request**, there shall be exactly one instance of **cc_design_date_and_-time_assignment** that contains the instance of **start_request** in its set of **items**.

5.2.5.14 start_work_requires_approval

The **start_work_requires_approval** rule specifies that each instance of **start_work** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every design initiation to have an approval.

EXPRESS specification:

```
*)
RULE start_work_requires_approval FOR
  (start_work, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (sw <* start_work |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       sw IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

start_work: identifies the set of all instances of **start_work** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **start_work**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **start_work** in its set of **items**.

5.2.5.15 start_work_requires_date_time

The **start_work_requires_date_time** rule specifies that each instance of a **start_work** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **start_-work** to have a start date indicating the date on which work on the design is initiated.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**.

EXPRESS specification:

```
*)
RULE start_work_requires_date_time FOR
  (start_work, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (sw <* start_work |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       (sw IN ccdta.items)
       AND (ccdta.role.name = 'start_date'))) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

start_work: identifies the set of all instances of **start_work** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **start_work**, there shall be exactly one instance of **cc_design_date_and_time_-assignment** that contains the instance of **start_work** in its set of **items** where the role name is "start_date".

5.2.5.16 restrict_action_request_status

The **restrict_action_request_status** rule specifies the permitted values for the status of a **action_request**.

EXPRESS specification:

```
*)
RULE restrict_action_request_status FOR (action_request_status);
WHERE
  WR1: SIZEOF (QUERY (ars <* action_request_status |
       NOT (ars.status IN ['proposed', 'in_work', 'issued', 'hold']))) = 0;
END_RULE;
(*
```

Argument definitions:

action_request_status: identifies the set of all instances of **action_request_status** entities.

Formal propositions:

WR1: For each instance of the **action_request_status** entity, the status attribute shall have a value of "proposed", "in_work", "issued", or "hold".

Attribute value definitions:

proposed: the **versioned_action_request** has been completed and is awaiting review and authorization.

in_work: the **versioned_action_request** is being developed for possible inclusion in the design.

issued: the **versioned_action_request** has been authorized for inclusion in the design.

hold: the **versioned_action_request** has been reviewed and not authorized to be included in the design.

5.2.5.17 versioned_action_request_requires_status

The **versioned_action_request_requires_status** rule specifies that each instance of **versioned_action_-request** shall have exactly one status. The status of a **versioned_action_request** is defined by the **action_-request_status** entity.

EXPRESS specification:

```
*)
RULE versioned_action_request_requires_status FOR
  (versioned_action_request, action_request_status);
WHERE
  WR1: SIZEOF (QUERY (ar <* versioned_action_request |
       NOT (SIZEOF (QUERY (ars <* action_request_status |
       ar :=: ars.assigned_request)) = 1))) = 0;
END_RULE;
(*
```

Argument definitions:

versioned_action_request: identifies the set of all instances of **versioned_action_request**.

action_request_status: identifies the set of all instances of **action_request_status**.

Formal proposition:

WR1: For each instance of **versioned_action_request** there shall be exactly one instance of **action_-request_status** that contains an **assigned_request** attribute value equal to that instance of **versioned_-action_request**.

5.2.5.18 versioned_action_request_requires_solution

The **versioned_action_request_requires_solution** rule specifies that each instance of **versioned_action_-request** shall have one or more solutions suggested for it. A solution for a **versioned_action_request** is defined by the **action_request_solution** entity.

EXPRESS specification:

```
*)
RULE versioned_action_request_requires_solution FOR
  (versioned_action_request, action_request_solution);
WHERE
```

```
  WR1: SIZEOF (QUERY (ar <* versioned_action_request |
       NOT (SIZEOF (QUERY (ars <* action_request_solution |
       ar :=: ars.request)) >= 1))) = 0;
END_RULE;
(*
```

Argument definitions:

versioned_action_request: identifies the set of all instances of **versioned_action_request**.

action_request_solution: identifies the set of all instances of **action_request_solution**.

Formal proposition:

WR1: For each instance of **versioned_action_request** there shall be exactly one instance of **action_-request_solution** that contains a **request** attribute value equal to that instance of **versioned_action_request**.

5.2.5.19 unique_version_change_order_rule

The **unique_version_change_order_rule** calls a function that returns true if a **change** updates more than one **product_definition_formation**, and each updated **product_definition_formation** is a version of a different product. This rule specifies that a single **change** shall not change more than one version of a single **product**, but may change more than one **product_definition_formation** if each **product_definition_-formation** references a different **product**.

EXPRESS specification:

```
*)
RULE unique_version_change_order_rule FOR (change);
WHERE
  WR1: SIZEOF (QUERY (c <* change |
       NOT (unique_version_change_order (c.assigned_action)))) = 0;
END_RULE;
(*
```

Argument definitions:

change: identifies the set of all instances of **change** entities.

Formal propositions:

WR1: For each instance of **change**, the **unique_version_change_order** function shall return a value of true.

5.2.5.20 product_requires_version

The **product_requires_version** rule specifies that each instance of **product** shall be referenced by at least one instance of **product_definition_formation**. This rule enforces the requirement for every product to have one or more versions.

EXPRESS specification:

```
*)
RULE product_requires_version FOR (product, product_definition_formation);
WHERE
  WR1: SIZEOF (QUERY (prod <* product |
       NOT (SIZEOF (QUERY (pdf <* product_definition_formation |
       prod :=: pdf.of_product )) >= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product: identifies the set of all instances of **product** entities.

product_definition_formation: identifies the set of all instances of **product_definition_formation** entities.

Formal propositions:

WR1: For each instance of **product**, there shall be one or more instances of **product_definition_formation** that contains an **of_product** attribute value equal to that instance of **product**.

5.2.5.21 product_requires_person_organization

The **product_requires_person_organization** rule specifies that each instance of **product** shall be referenced by an instance of **cc_design_person_and_organization_assignment**. This rule enforces the requirement for every product to have an design_owner.

EXPRESS specification:

```
*)
RULE product_requires_person_organization FOR
  (product, cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (prod <* product |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       prod IN ccdpoa.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product: identifies the set of all instances of **product** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **product**, there shall be an instance of **cc_design_person_and_organization_-assignment** that contains an **items** attribute value equal to that instance of **product**.

NOTE - The role which is specified for the **person_and_organization** that is assigned to the **product** is "design_owner". This role association is specified formally in the function defined in 5.2.6.2.

5.2.5.22 product_version_requires_approval

The **product_version_requires_approval** rule specifies that each instance of **product_definition_-formation** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every version of a design to have an approval.

EXPRESS specification:

```
*)
RULE product_version_requires_approval FOR  (product_definition_formation,
  cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (pdf <* product_definition_formation |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
```

```
       pdf IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition_formation: identifies the set of all instances of **product_definition_formation** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **product_definition_formation**, there shall be exactly one instance of **cc_-design_approval** that contains the instance of **product_definition_formation** in its set of **items**.

5.2.5.23 product_version_requires_person_organization

The **product_version_requires_person_organization** rule specifies that every **product_definition_-formation** shall be referenced by exactly one **cc_design_person_and_organization_assignment** in the role of creator and one or more **cc_design_person_and_organization_assignment** in the role of either part_-supplier or design_supplier. This rule specifies the need for every **product_definition_formation** to have a creator and supplier who are responsible for the creation or delivery of the particular version of the design. The meanings of creator, part_supplier, and design_supplier are found in 5.2.5.43 as part of the definition of the **restrict_person_organization_role** rule.

> NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**. See the function definition in 5.2.6.2.

EXPRESS specification:

```
*)
RULE product_version_requires_person_organization FOR
  (product_definition_formation,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (pdf <* product_definition_formation |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       (pdf IN ccdpoa.items) AND (ccdpoa.role.name = 'creator'))) = 1 )))
       = 0;
```

```
  WR2: SIZEOF (QUERY (pdf <* product_definition_formation |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       (pdf IN ccdpoa.items) AND
       (ccdpoa.role.name IN ['design_supplier', 'part_supplier']))) >= 1)))
       = 0;
END_RULE;
(*
```

Argument definitions:

product_definition_formation: identifies the set of all instances of **product_definition_formation** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **product_definition_formation**, there shall be exactly one instance of **cc_-design_person_and_organization_assignment** that contains the instance of **product_definition_-formation** in its set of **items** and its **role** attribute references a **person_and_organization_role** that has a value of 'creator' in its **name** attribute.

WR2: For each instance of **product_definition_formation**, there shall be at least one instance of **cc_-design_person_and_organization_assignment** that contains the instance of **product_definition_-formation** in its set of **items** and its **role** attribute references a **person_and_organization_role** that has a value of either 'design_supplier' or 'part_supplier' in its **name** attribute.

5.2.5.24 product_version_requires_security_classification

The **product_version_requires_security_classification** rule specifies that each instance of **product_-definition_formation** shall be referenced by exactly one instance of **cc_design_security_classification**. This rule enforces the requirement for every version of a design to have a security classification.

EXPRESS specification:

```
*)
RULE product_version_requires_security_classification FOR
  (product_definition_formation, cc_design_security_classification);
WHERE
  WR1: SIZEOF (QUERY (pdf <* product_definition_formation |
       NOT (SIZEOF (QUERY (ccdsc <* cc_design_security_classification |
```

```
       pdf IN ccdsc.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition_formation: identifies the set of all instances of **product_definition_formation** entities.

cc_design_security_classification: identifies the set of all instances of **cc_design_security_classification** entities.

Formal propositions:

WR1: For each instance of **product_definition_formation**, there shall be exactly one instance of **cc_-design_security_classification** that contains the instance of **product_definition_formation** in its set of **items**.

5.2.5.25 product_definition_requires_person_organization

The **product_definition_requires_person_organization** rule specifies that every **product_definition** shall be referenced by exactly one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **product_definition** to have a creator who is responsible for the creation of the particular definition of the design. The meaning of creator is found in the **person_and_organization_assignment** entity's **role** attribute.

> NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**.

EXPRESS specification:

```
*)
RULE product_definition_requires_person_organization FOR
  (product_definition,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (pd <* product_definition |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       pd IN ccdpoa.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition: identifies the set of all instances of **product_definition** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **product_definition**, exactly one instance of **cc_design_person_and_-organization_assignment** shall contain the instance of **product_definition** in its set of **items**.

5.2.5.26 product_definition_requires_approval

The **product_definition_requires_approval** rule specifies that each instance of **product_definition** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every definition of a design to have an approval.

EXPRESS specification:

```
*)
RULE product_definition_requires_approval FOR
  (product_definition, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (pd <* product_definition |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       pd IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition: identifies the set of all instances of **product_definition** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **product_definition**, exactly one instance of **cc_design_approval** shall contain the instance of **product_definition** in its set of **items**.

5.2.5.27 product_definition_requires_date_time

The **product_definition_requires_date_time** rule specifies that each instance of a **product_definition** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **product_definition** to have a date indicating the date on which the definition of the product was created.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**.

EXPRESS specification:

```
*)
RULE product_definition_requires_date_time FOR
  (product_definition, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (pd <* product_definition |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       pd IN ccdta.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition: identifies the set of all instances of **product_definition** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **product_definition**, exactly one instance of **cc_design_date_and_time_-assignment** shall contain the instance of **product_definition** in its set of **items**.

5.2.5.28 certification_requires_approval

The **certification_requires_approval** rule specifies that each instance of **certification** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every certification of a supplier of either a design or a part to have an approval.

EXPRESS specification:

```
*)
RULE certification_requires_approval FOR (certification,
  cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (cert <* certification |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       cert IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

certification: identifies the set of all instances of **certification** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **certification**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **certification** its set of **items**.

5.2.5.29 restrict_certification_type

The **restrict_certification_type** rule specifies that **certification** may only be given for either a "part_-supplier" or "design_supplier".

EXPRESS specification:

```
*)
RULE restrict_certification_type FOR (certification_type);
WHERE
  WR1: SIZEOF (QUERY (ct <* certification_type |
       NOT (ct.description IN ['design_supplier', 'part_supplier']))) = 0;
```

```
END_RULE;
(*
```

Argument definitions:

certification_type: identifies the set of all instances of **certification_type** entities.

Formal propositions:

WR1: For each instance of **certification_type**, the value of the **kind** attribute shall be "design_supplier" or "part_supplier".

Attribute value definitions:

design_supplier: identifies a supplier of a design for a part.

part_supplier: identifies a supplier of a part.

5.2.5.30 certification_requires_date_time

The **certification_requires_date_time** rule specifies that each instance of a **certification** shall be referenced by exactly one **cc_design_date_and_time_assignment**. This rule enforces the requirement for every **certification** to have a date indicating the date on which the certification is effective.

> NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**. See the function definition in 5.2.6.3.

EXPRESS specification:

```
*)
RULE certification_requires_date_time FOR
  (certification, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (cert <* certification |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       cert IN ccdta.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

certification: identifies the set of all instances of **certification** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **certification**, exactly one instance of **cc_design_date_and_time_assignment** shall contain the instance of **certification** in its set of **items**.

5.2.5.31 approvals_are_assigned

The **approvals_are_assigned** rule specifies that each instance of **approval** shall be assigned by at least one instance of approval_assignment entity.

EXPRESS specification:

```
*)
RULE approvals_are_assigned FOR
  (approval, approval_assignment);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (aa <* approval_assignment |
       app :=: aa.assigned_approval )) >= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

approval: identifies the set of all instances of **approval** entities.

approval_assignment: identifies the set of all instances of **approval_assignment** entities.

Formal propositions:

WR1: For each instance of **approval**, there shall be one or more instances of **approval_assignment** which contains the instance of **approval** as its **assigned_approval** attribute.

5.2.5.32 approval_requires_approval_person_organization

The **approval_requires_approval_person_organization** specifies that each instance of **approval** shall have at least one **approval_person_organization** referencing it. This rule enforces the requirement for an approval to be authorized by one or more people within their organizations.

EXPRESS specification:

```
*)
RULE approval_requires_approval_person_organization FOR
  (approval, approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (apo <* approval_person_organization |
       app :=: apo.authorized_approval )) >= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

approval: identifies the set of all instances of **approval** entities.

approval_person_organization: identifies the set of all instances of **approval_person_organization** entities.

Formal propositions:

WR1: For each instance of **approval**, there shall be one or more instances of **approval_person_-organization** which contains the instance of **approval** as its **authorized_approval** attribute.

5.2.5.33 approval_requires_approval_date_time

The **approval_requires_approval_date_time** rule specifies that each instance of **approval** shall be referenced by exactly one **approval_date_time**. This rule enforces the requirement for every approval to have a date on which the approval obtained its specified status.

EXPRESS specification:

```
*)
RULE approval_requires_approval_date_time FOR (approval,
  approval_date_time);
WHERE
```

```
  WR1: SIZEOF (QUERY (app <* approval |
       NOT (SIZEOF (QUERY (adt <* approval_date_time |
       app :=: adt.dated_approval )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

approval: identifies the set of all instances of **approval** entities.

approval_date_time: identifies the set of all instances of **approval_date_time** entities.

Formal propositions:

WR1: For each instance of **approval**, there shall be exactly one instance of **approval_date_time** which contains the instance of **approval** as its **dated_approval** attribute.

5.2.5.34 restrict_approval_status

The **restrict_approval_status** rule specifies that the only values of **approval_status** permitted shall be "approved", "not_yet_approved", "disapproved" or "withdrawn".

EXPRESS specification:

```
*)
RULE restrict_approval_status FOR (approval_status);
WHERE
  WR1: SIZEOF (QUERY (ast <* approval_status |
       NOT (ast.name IN
       ['approved', 'not_yet_approved', 'disapproved', 'withdrawn']))) = 0;
END_RULE;
(*
```

Argument definitions:

approval: identifies the set of all instances of **approval** entities.

Formal propositions:

WR1: For each instance of **approval**, the value of the **status** attribute shall be either "approved", "not_yet_-approved", "disapproved" or "withdrawn".

Attribute value definitions:

approved: specifies that the required authorizations have been obtained for a particular role of approval for a piece of product data.

not_yet_approved: specifies that the required authorizations are being awaited for a particular role of approval for a piece of product data.

disapproved: specifies that the required authorizations have been denied for a particular role of approval for a piece of product data.

withdrawn: specifies that the required authorizations have been revoked for a particular role of approval for a piece of product data.

5.2.5.35 contract_requires_approval

The **contract_requires_approval** rule specifies that each instance of **contract** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every contract under which a design is developed is to have an approval.

EXPRESS specification:

```
*)
RULE contract_requires_approval FOR (contract,
  cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (c <* contract |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       c IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

contract: identifies the set of all instances of **contract** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **contract**, exactly one instance of **cc_design_approval** shall exist which references the instance of **contract** in its set of **items.**

5.2.5.36 contract_requires_person_organization

The **contract_requires_person_organization** rule specifies that every **contract** shall be referenced by exactly one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **contract** to have a contractor who is responsible for the maintenance of information about the contract. The meaning of contractor is found in the **person_and_organization_assignment** entity's **role** attribute.

NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**. See the function definition in 5.2.6.2.

EXPRESS specification:

```
*)
RULE contract_requires_person_organization FOR
  (contract, cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (c <* contract |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       c IN ccdpoa.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

contract: identifies the set of all instances of **contract** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **contract**, there shall be exactly one instance of **cc_design_person_and_-organization_assignment** that contains the instance of **contract** in its set of **items**.

5.2.5.37 restrict_contract_type

The **restrict_contract_type** rule specifies the permitted types of contracts. This rule enforces the requirement for the types of contract to be either "fixed_price" or "cost_plus".

EXPRESS specification:

```
*)
RULE restrict_contract_type FOR (contract_type);
WHERE
  WR1: SIZEOF (QUERY (ct <* contract_type |
       NOT (ct.description IN ['fixed_price', 'cost_plus']))) = 0;
END_RULE;
(*
```

Argument definitions:

contract_type: identifies the set of all instances of **contract_type** entities.

Formal propositions:

WR1: For each instance of **contract_type**, the **kind** attribute shall contain a value of "fixed_price" or "cost_plus".

Attribute value definitions:

fixed_price: identifies a contract under which the remuneration for the performance of the contract has been set.

cost_plus: identifies a contract under which the remuneration for the performance of the contract is a set amount above the cost to the contracting organization.

5.2.5.38 security_classification_requires_approval

The **security_classification_requires_approval** rule specifies that each instance of **security_classification** shall be referenced by exactly one instance of **cc_design_approval**. This rule enforces the requirement for every security classification assigned to a design is to have an approval.

EXPRESS specification:

```
*)
RULE security_classification_requires_approval FOR
  (security_classification, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (sc <* security_classification |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       sc IN ccda.items )) = 1 ))) = 0;
```

```
END_RULE;
(*
```

Argument definitions:

security_classification: identifies the set of all instances of **security_classification** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **security_classification**, there shall be exactly one instance of **cc_design_-approval** that contains the instance of **security_classification** in its set of **items**.

5.2.5.39 security_classification_requires_person_organization

The **security_classification_requires_person_organization** rule specifies that every **security_-classification** shall be referenced by exactly one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **security_classification** to have a classification officer who is responsible for issuing the security classification. The meaning of classification officer is found in the **person_and_-organization_assignment** entity's **role** attribute.

> NOTE - The coordination of the different role values with the assignment of **person_and_organization_-assignment** to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**. See the function definition in 5.2.6.2.

EXPRESS specification:

```
*)
RULE security_classification_requires_person_organization FOR
  (security_classification,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (sc <* security_classification |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       sc IN ccdpoa.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

security_classification: identifies the set of all instances of **security_classification** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **security_classification**, there shall be exactly one instance of **cc_design_-person_and_organization_assignment** that contains the instance of **security_classification** in its set of **items**.

5.2.5.40 security_classification_requires_date_time

The **security_classification_requires_date_time** rule specifies that each instance of a **security_-classification** shall be referenced by exactly one **cc_design_date_and_time_assignment** that has a role of "classification_date". This rule enforces the requirement for every **security_classification** to have a date indicating the date on which the security classification is effective.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**. See the function definition in 5.2.6.3.

EXPRESS specification:

```
*)
RULE security_classification_requires_date_time FOR
  (security_classification, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (sc <* security_classification |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       (sc IN ccdta.items) AND
       ('classification_date' = ccdta.role.name))) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

security_classification: identifies the set of all instances of **security_classification** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **security_classification**, there shall be exactly one instance of **cc_design_date_-and_time_assignment** that contains the instance of **security_classification** in its set of **items** and its **the_-role** attribute references an instance of **date_and_time_role** that has a **name** attribute with the value of "classification_date".

5.2.5.41 security_classification_optional_date_time

The **security_classification_optional_date_time** rule specifies that each instance of a **security_-classification** may be referenced by zero or one **cc_design_date_and_time_assignment** that has a role of "declassification_date". This rule enforces the requirement that every **security_classification** may have a date indicating the date on which the security classification expires.

NOTE - The coordination of the different role values with the assignment of **date_time_assignment** to different entities is specified in the **cc_design_date_and_time_correlation** function. This function is invoked locally to **cc_design_date_and_time_assignment**. See the function definition in 5.2.6.3.

EXPRESS specification:

```
*)
RULE security_classification_optional_date_time FOR
  (security_classification, cc_design_date_and_time_assignment);
WHERE
  WR1: SIZEOF (QUERY (sc <* security_classification |
       NOT (SIZEOF (QUERY (ccdta <* cc_design_date_and_time_assignment |
       (sc IN ccdta.items) AND
       ('declassification_date' = ccdta.role.name))) <= 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

security_classification: identifies the set of all instances of **security_classification** entities.

cc_design_date_and_time_assignment: identifies the set of all instances of **cc_design_date_and_time_-assignment** entities.

Formal propositions:

WR1: For each instance of **security_classification**, there may be zero or one instance of **cc_design_date_-and_time_assignment** that contains the instance of **security_classification** in its set of **items** and its **the_-role** attribute references an instance of **date_and_time_role** that has a **name** attribute with the value of "declassification_date".

5.2.5.42 restrict_security_classification_level

The **restrict_security_classification_level** rule specifies the permitted levels of security. This rule enforces the requirement for the levels of security to be "unclassified", "classified", "proprietary", "confidential", "secret", or "top_secret".

EXPRESS specification:

```
*)
RULE restrict_security_classification_level FOR
  (security_classification_level);
WHERE
  WR1: SIZEOF (QUERY (scl <* security_classification_level |
       NOT (scl.name IN ['unclassified', 'classified', 'proprietary',
       'confidential',  'secret', 'top_secret']))) = 0;
END_RULE;
(*
```

Argument definitions:

security_classification_level: identifies the set of all instances of **security_classification_level** entities.

Formal propositions:

WR1: For each instance of **security_classification_level**, the **name** attribute shall contain a value of "unclassified", "classified", "proprietary", "confidential", "secret" or "top_secret".

Attribute value definitions:

unclassified: identifies the classification level for which no security is necessary.

classified: identifies the classification level for which security is necessary, but the classification details are not given.

proprietary: identifies the classification level for which the disclosure of information about the part or the design of the part would risk an organization's market or competitive advantage.

confidential: identifies the classification level for which the disclosure of information about the part or the design of the part would cause damage to national or organizational security.

secret: identifies the classification level for which the disclosure of information about the part or the design of the part would cause serious damage to national or organizational security.

top_secret: identifies the classification level for which the disclosure of information about the part or the design of the part would cause exceptionally grave damage to national or organizational security.

5.2.5.43 restrict_person_organization_role

The **restrict_person_organization_role** rule specifies the permitted roles for **person_and_organization** entities. This rule enforces the requirement for the roles of **person_and_organization** entities to be "request_recipient", "initiator", "part_supplier", "design_supplier", "configuration_manager", "contractor", "classification_officer", "creator" or "design_owner".

EXPRESS specification:

```
*)
RULE restrict_person_organization_role FOR
  (person_and_organization_role);
WHERE
  WR1: SIZEOF (QUERY (por <* person_and_organization_role |
       NOT (por.name IN ['request_recipient', 'initiator', 'part_supplier',
       'design_supplier', 'configuration_manager', 'contractor',
       'classification_officer', 'creator', 'design_owner']))) = 0;
END_RULE;
(*
```

Argument definitions:

person_organization_role: identifies the set of all instances of **person_organization_role** entities.

Formal propositions:

WR1: For each instance of **person_organization_role**, the name attribute shall have a value of "request_-recipient", "initiator", "part_supplier", "design_supplier", "configuration_manager", "contractor", "classification_officer", "creator" or "design_owner".

Attribute value definitions:

request_recipient: identifies a person within an organization who is responsible for receiving a **change_-request** or **start_request** and taking action based on the request.

initiator: identifies a person within an organization who is responsible for creating the **change_request** or **start_request**.

part_supplier: identifies a person within an organization who is responsible for supplying a part.

design_supplier: identifies a person within an organization who is responsible for supplying the design of a part.

configuration_manager: identifies a person within an organization who is responsible for assigning the configuration information about a design.

contractor: identifies a person within an organization who is responsible for the information pertaining to a contract that applies to a design.

classification_officer: identifies a person within an organization who is responsible for the classification and declassification of parts.

creator: identifies a person within an organization who is responsible for the creation of a particular **product_definition_formation** or **product_definition**.

design_owner: identifies a person within an organization who is responsible for the design of all aspects of a **product**.

5.2.5.44 restrict_date_time_role

The **restrict_date_time_role** rule specifies the permitted roles for **date_and_time** entities. This rule enforces the requirement for the roles of **date_and_time** entities to be "creation_date", "request_date", "release_date", "start_date", "contract_date", "certification_date", "sign_off_date", "classification_date", or "declassification_date".

EXPRESS specification:

```
*)
RULE restrict_date_time_role FOR (date_time_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* date_time_role |
       NOT (dtr.name IN ['creation_date', 'request_date', 'release_date',
```

```
        'start_date', 'contract_date', 'certification_date',
        'sign_off_date', 'classification_date',
      'declassification_date']))) = 0;
END_RULE;
(*
```

Argument definitions:

date_time_role: identifies the set of all instances of **date_time_role** entities.

Formal propositions:

WR1: For each instance of **date_time_role**, the name attribute shall have the value of "creation_date", "request_date", "release_date", "start_date", "contract_date", "certification_date", "sign_off_date", "classification_date", or "declassification_date".

Attribute value definitions:

creation_date: identifies the date and time when either a version of a design or a definition of a design comes into existence.

request_date: identifies the date and time given when work is requested for a design.

release_date: identifies the date and time when a design is initially released or a change is incorporated into a design.

start_date: identifies the date and time when work is begun for either a new design or a change to an existing design.

contract_date: identifies the date and time when a contract for a design goes into effect.

certification_date: identifies the date and time when a certification goes into effect.

sign_off_date: identifies the date and time when an authorization was given to an approval.

classification_date: identifies the date and time when a security classification goes into effect.

declassification_date: identifies the date and time when a security classification goes out of effect.

5.2.5.45 restrict_document_type

The **restrict_document_type** rule specifies the permitted types of documents. This rule enforces the requirement for the types of documents to be "material_specification", "process_specification", "design_-specification" or "surface_finish_specification".

EXPRESS specification:

```
*)
RULE restrict_document_type FOR (document_type);
WHERE
  WR1: SIZEOF (QUERY (dt <* document_type |
       NOT (dt.product_data_type IN ['material_specification',
       'process_specification', 'design_specification',
       'surface_finish_specification', 'cad_filename', 'drawing']))) = 0;
END_RULE;
(*
```

Argument definitions:

document_type: identifies the set of all instances of **document_type** entities.

Formal propositions:

WR1: For each instance of **document_type**, the **product_data_type** attribute shall have the value of "material_specification", "process_specification", "design_specification", "surface_finish_specification", "cad_filename", or "drawing".

Attribute value definitions:

material_specification: identifies a type of document that specifies properties which are applicable to raw material, mixtures or semi-fabricated material which are used in the fabrication of a part.

process_specification: identifies a type of document that specifies a service to be performed on a product or material.

design_specification: identifies a type of document that sets the design requirements of a part.

surface_finish_specification: identifies a type of document that specifies properties which are applicable to surface textures or protective coatings for either an in process or completed part.

cad_filename: identifies a type of document that is represented electronically as a file from a Computer Aided Design system.

drawing: identifies a type of document that contains the draughting representation of the design for a part or parts.

5.2.5.46 document_to_product_definition

The **document_to_product_definition** rule specifies the permitted association between documents which are grouped together using **document_relationship** and **product_definition** entities. This rule enforces the requirement that only **product_definition** entities may have groups of documents related to them.

EXPRESS specification:

```
*)
RULE document_to_product_definition FOR
  (cc_design_specification_reference);
WHERE
  WR1: SIZEOF (QUERY (sp <* cc_design_specification_reference |
       NOT ((((('CONFIG_CONTROL_DESIGN.DOCUMENT_RELATIONSHIP.' +
       'RELATING_DOCUMENT' IN
       ROLESOF (sp\document_reference.assigned_document)) AND
       (SIZEOF (QUERY (it <* sp.items |
       NOT('CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION' IN
       TYPEOF (it)))) = 0)))
       OR
       (NOT ('CONFIG_CONTROL_DESIGN.DOCUMENT_RELATIONSHIP.' +
       'RELATING_DOCUMENT' IN
       ROLESOF (sp\document_reference.assigned_document)))))) = 0;
END_RULE;
(*
```

Argument definitions:

cc_design_specification_reference: identifies the set of all instances of **cc_design_specification_-reference** entities.

product_definition: identifies the set of all instances of **product_definition** entities.

Formal propositions:

WR1: For each instance of **cc_design_specification_reference**, if the referenced **document** is used as the **relating_document** in a **document_relationship** then each of the elements in the set of **items** shall be a **product_definition** entity.

5.2.5.47 as_required_quantity

The **as_required_quantity** rule specifies the use of the **descriptive_measure** type in the **measure_with_-unit** entity. The value of the **descriptive_measure** STRING type shall always be "as_required". This rule enforces the requirement for the specification of the use of an amount to be as required.

EXPRESS specification:

```
*)
RULE as_required_quantity FOR (measure_with_unit);
WHERE
  WR1: SIZEOF (QUERY (m <* measure_with_unit |
       ('CONFIG_CONTROL_DESIGN.DESCRIPTIVE_MEASURE' IN
       TYPEOF (m.value_component)) AND
       (NOT (m.value_component = 'as_required')))) = 0;
END_RULE;
(*
```

Argument definitions:

measure_with_unit: identifies the set of all instances of **measure_with_unit** entities.

Formal propositions:

WR1: For each instance of **measure_with_unit**, if the type of the attribute **value** is **descriptive_measure** then the value of the attribute shall be "as_required".

5.2.5.48 global_unit_assignment

The **global_unit_assignment** rule specifies the units that shall be defined for a **global_unit_assigned_-context**. The rule states that every **global_unit_assigned_context** shall have exactly three elements in its set of **units**, and that every **global_unit_assigned_context** shall contain units of length, plane angle and solid angle.

EXPRESS specification:

```
*)
RULE global_unit_assignment FOR (global_unit_assigned_context);
WHERE
  WR1: SIZEOF (QUERY (guac <* global_unit_assigned_context |
       NOT (SIZEOF (guac.units) = 3))) = 0;
  WR2: SIZEOF (QUERY (guac <* global_unit_assigned_context |
       NOT ((SIZEOF (QUERY (u <* guac.units |
       'CONFIG_CONTROL_DESIGN.LENGTH_UNIT' IN TYPEOF (u))) = 1) AND
       (SIZEOF (QUERY (u <* guac.units |
       'CONFIG_CONTROL_DESIGN.PLANE_ANGLE_UNIT' IN TYPEOF (u))) = 1) AND
       (SIZEOF (QUERY (u <* guac.units |
       'CONFIG_CONTROL_DESIGN.SOLID_ANGLE_UNIT' IN TYPEOF (u))) = 1
       )))) = 0;
END_RULE;
(*
```

Argument definitions:

global_unit_assigned_context: identifies the set of all instances of **global_unit_assigned_context** entities.

Formal propositions:

WR1: For each instance of **global_unit_assigned_context**, the set of **units** shall contain exactly 3 elements.

WR2: For each instance of **global_unit_assigned_context**, the first element of the set of **units** shall be a **length_unit**, the second element of the set of **units** shall be a **plane_angle_unit** and the third element of the set of **units** shall be a **solid_angle_unit**.

5.2.5.49 subtype_mandatory_action

The **subtype_mandatory_action** rule specifies that all **action** entities shall be an **directed_action** entity.

EXPRESS specification:

```
*)
RULE subtype_mandatory_action FOR
  (action);
WHERE
  WR1: SIZEOF (QUERY (act <* action |
       NOT ('CONFIG_CONTROL_DESIGN.DIRECTED_ACTION' IN
```

```
       TYPEOF (act)))) = 0;
END_RULE;
(*
```

Argument definitions:

action: identifies the set of all instances of **action** entities to be constrained.

Formal propositions:

WR1: Each instance of **action** shall be a **directed_action**.

5.2.5.50 subtype_mandatory_product_definition_formation

The **subtype_mandatory_product_definition_formation** rule specifies that all **product_definition_-formation** entities shall be **product_definition_formation_with_specified_source** entities.

EXPRESS specification:

```
*)
RULE subtype_mandatory_product_definition_formation FOR
  (product_definition_formation);
WHERE
  WR1: SIZEOF (QUERY (pdf <* product_definition_formation |
       NOT ('CONFIG_CONTROL_DESIGN.' +
       'PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE' IN
       TYPEOF (pdf)))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition_formation: identifies the set of all instances of **product_definition_formation** entities to be constrained.

Formal propositions:

WR1: Each instance of **product_definition_formation** shall be a **product_definition_formation_with_-specified_source**.

5.2.5.51 dependent_instantiable_date

The **dependent_instantiable_date** rule specifies that all instances of **date** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_date FOR (date);
WHERE
  WR1: SIZEOF (QUERY (dt <* date |
       NOT (SIZEOF (USEDIN (dt, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

date: identifies the set of all instances of **date**.

Formal proposition:

WR1: For each instance of **date**, there shall be a reference to the **date** instance from an attribute of another entity.

5.2.5.52 dependent_instantiable_shape_representation

The **dependent_instantiable_shape_representation** rule specifies that all instances of **shape_-representation** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_shape_representation FOR
  (shape_representation);
WHERE
  WR1: SIZEOF (QUERY (sr <* shape_representation |
       NOT (SIZEOF (USEDIN (sr, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

shape_representation: identifies the set of all instances of **shape_representation**.

Formal proposition:

WR1: For each instance of **shape_representation**, there shall be a reference to the **shape_representation** instance from an attribute of another entity.

5.2.5.53 dependent_instantiable_named_unit

The **dependent_instantiable_named_unit** rule specifies that all instances of **named_unit** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_named_unit FOR (named_unit);
WHERE
  WR1: SIZEOF (QUERY (nu <* named_unit |
       NOT (SIZEOF (USEDIN (nu, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

named_unit: identifies the set of all instances of **named_unit**.

Formal proposition:

WR1: For each instance of **named_unit**, there shall be a reference to the **named_unit** instance from an attribute of another entity.

5.2.5.54 dependent_instantiable_representation_item

The **dependent_instantiable_representation_item** rule specifies that all instances of **representation_item** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_representation_item FOR (representation_item);
```

```
WHERE
  WR1: SIZEOF (QUERY (ri <* representation_item |
       NOT (SIZEOF (USEDIN (ri, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

representation_item: identifies the set of all instances of **representation_item**.

Formal proposition:

WR1: For each instance of **representation_item**, there shall be a reference to the **representation_item** instance from an attribute of another entity.

5.2.5.55 dependent_instantiable_date_time_role

The **dependent_instantiable_date_time_role** rule specifies that all instances of **date_time_role** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_date_time_role FOR (date_time_role);
WHERE
  WR1: SIZEOF (QUERY (dtr <* date_time_role |
       NOT (SIZEOF (USEDIN (dtr, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

date_time_role: identifies the set of all instances of **date_time_role**.

Formal proposition:

WR1: For each instance of **date_time_role**, there shall be a reference to the **date_time_role** instance from an attribute of another entity.

5.2.5.56 dependent_instantiable_person_and_organization_role

The **dependent_instantiable_person_and_organization_role** rule specifies that all instances of **person_-and_organization_role** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_person_and_organization_role FOR
  (person_and_organization_role);
WHERE
  WR1: SIZEOF (QUERY (poar <* person_and_organization_role |
       NOT (SIZEOF (USEDIN (poar, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

person_and_organization_role: identifies the set of all instances of **person_and_organization_role**.

Formal proposition:

WR1: For each instance of **person_and_organization_role**, there shall be a reference to the **person_and_-organization_role** instance from an attribute of another entity.

5.2.5.57 dependent_instantiable_action_directive

The **dependent_instantiable_action_directive** rule specifies that all instances of **action_directive** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_action_directive FOR (action_directive);
WHERE
  WR1: SIZEOF (QUERY (ad <* action_directive |
       NOT (SIZEOF (USEDIN (ad, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

action_directive: identifies the set of all instances of **action_directive**.

Formal proposition:

WR1: For each instance of **action_directive**, there shall be a reference to the **action_directive** instance from an attribute of another entity.

5.2.5.58 dependent_instantiable_security_classification_level

The **dependent_instantiable_security_classification_level** rule specifies that all instances of **security_-classification_level** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_security_classification_level FOR
  (security_classification_level);
WHERE
  WR1: SIZEOF (QUERY (scl <* security_classification_level |
       NOT (SIZEOF (USEDIN (scl, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

security_classification_level: identifies the set of all instances of **security_classification_level**.

Formal proposition:

WR1: For each instance of **security_classification_level**, there shall be a reference to the **security_-classification_level** instance from an attribute of another entity.

5.2.5.59 dependent_instantiable_approval_status

The **dependent_instantiable_approval_status** rule specifies that all instances of **approval_status** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_approval_status FOR (approval_status);
WHERE
  WR1: SIZEOF (QUERY (ast <* approval_status |
       NOT (SIZEOF (USEDIN (ast, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

approval_status: identifies the set of all instances of **approval_status**.

Formal proposition:

WR1: For each instance of **approval_status**, there shall be a reference to the **approval_status** instance from an attribute of another entity.

5.2.5.60 dependent_instantiable_document_type

The **dependent_instantiable_document_type** rule specifies that all instances of **document_type** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_document_type FOR (document_type);
WHERE
  WR1: SIZEOF (QUERY (dt <* document_type |
       NOT (SIZEOF (USEDIN (dt, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

document_type: identifies the set of all instances of **document_type**.

Formal proposition:

WR1: For each instance of **document_type**, there shall be a reference to the **document_type** instance from an attribute of another entity.

5.2.5.61 dependent_instantiable_contract_type

The **dependent_instantiable_contract_type** rule specifies that all instances of **contract_type** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_contract_type FOR (contract_type);
WHERE
  WR1: SIZEOF (QUERY (ct <* contract_type |
       NOT (SIZEOF (USEDIN (ct, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

contract_type: identifies the set of all instances of **contract_type**.

Formal proposition:

WR1: For each instance of **contract_type**, there shall be a reference to the **contract_type** instance from an attribute of another entity.

5.2.5.62 dependent_instantiable_certification_type

The **dependent_instantiable_certification_type** rule specifies that all instances of **certification_type** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_certification_type FOR (certification_type);
WHERE
  WR1: SIZEOF (QUERY (ct <* certification_type |
       NOT (SIZEOF (USEDIN (ct, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

certification_type: identifies the set of all instances of **certification_type**.

Formal proposition:

WR1: For each instance of **certification_type**, there shall be a reference to the **certification_type** instance from an attribute of another entity.

5.2.5.63 product_concept_requires_configuration_item

The **product_concept_requires_configuration_item** rule specifies that every **product_concept** shall be referenced by at least one **configuration_item**. This rule enforces the requirement for every **product_-concept** have at least one **configuration_item** in it.

EXPRESS specification:

```
*)
RULE product_concept_requires_configuration_item FOR
  (product_concept, configuration_item);
WHERE
  WR1: SIZEOF (QUERY (pc <* product_concept |
       NOT (SIZEOF (QUERY (ci <* configuration_item |
       pc :=: ci.item_concept)) >=1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

product_concept: identifies the set of all instances of **product_concept** entities.

configuration_item: identifies the set of all instances of **configuration_item** entities.

Formal propositions:

WR1: For each instance of **product_concept**, there shall be at least one instance of **configuration_item** that contains the instance of **product_concept** as the value of its **item_concept** attribute.

5.2.5.64 configuration_item_requires_person_organization

The **configuration_item_requires_person_organization** rule specifies that every **configuration_item** shall be referenced by exactly one **cc_design_person_and_organization_assignment**. This rule specifies the need for every **configuration_item** to have a configuration manager who is responsible for it. The meaning of configuration_manager is found in the **person_organization_assignment** entity's **role** attribute.

NOTE - The coordination of the different role values with the assignment of **person_organization_-**

assignment to different entities is specified in the **cc_design_person_and_organization_correlation** function. This function is invoked locally to **cc_design_person_and_organization_assignment**. See the function definition in 5.2.6.2.

EXPRESS specification:

```
*)
RULE configuration_item_requires_person_organization FOR
  (configuration_item,
  cc_design_person_and_organization_assignment);
WHERE
  WR1: SIZEOF (QUERY (ci <* configuration_item |
       NOT (SIZEOF (QUERY (ccdpoa <*
       cc_design_person_and_organization_assignment |
       ci IN ccdpoa.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

configuration_item: identifies the set of all instances of **configuration_item** entities.

cc_design_person_and_organization_assignment: identifies the set of all instances of **cc_design_-person_and_organization_assignment** entities.

Formal propositions:

WR1: For each instance of **configuration_item**, there shall be exactly one instance of **cc_design_person_-organization_assignment** that contains the instance of **configuration_item** in its set of **items**.

5.2.5.65 subtype_mandatory_effectivity

The **subtype_mandatory_effectivity** rule specifies that each instance of the **effectivity** entities shall be a complex entity instance comprised of an instance of one of either **serial_numbered_effectivity**, **dated_-effectivity** or **lot_effectivity** entity and an instance of **configuration_effectivity**.

EXPRESS specification:

```
*)
RULE subtype_mandatory_effectivity FOR (effectivity);
WHERE
  WR1: SIZEOF (QUERY (eff <* effectivity |
```

```
       NOT ((SIZEOF (['CONFIG_CONTROL_DESIGN.SERIAL_NUMBERED_EFFECTIVITY',
       'CONFIG_CONTROL_DESIGN.LOT_EFFECTIVITY',
       'CONFIG_CONTROL_DESIGN.DATED_EFFECTIVITY'] *
       TYPEOF (eff)) = 1) AND
       ('CONFIG_CONTROL_DESIGN.CONFIGURATION_EFFECTIVITY' IN
       TYPEOF(eff))))) = 0;
END_RULE;
(*
```

Argument definitions:

effectivity: identifies the set of all instances of **effectivity** entities.

Formal propositions:

WR1: Each instance of **effectivity** shall be a **serial_numbered_effectivity**, **lot_effectivity** or **dated_-effectivity** and a **configuration_effectivity** instance.

5.2.5.66 effectivity_requires_approval

The **effectivity_requires_approval** rule specifies that each instance of a **effectivity** shall be referenced by exactly one **cc_design_approval**. This rule enforces the requirement for each **effectivity** to have an approval.

EXPRESS specification:

```
*)
RULE effectivity_requires_approval FOR
  (effectivity, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (eff <* effectivity |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       eff IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

effectivity: identifies the set of all instances of **effectivity** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **effectivity**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **effectivity** in its set of **items**.

5.2.5.67 configuration_item_requires_approval

The **configuration_item_requires_approval** rule specifies that each instance of a **configuration_item** shall be referenced by exactly one **cc_design_approval**. This rule enforces the requirement for each **configuration_item** to have an approval.

EXPRESS specification:

```
*)
RULE configuration_item_requires_approval FOR
  (configuration_item, cc_design_approval);
WHERE
  WR1: SIZEOF (QUERY (ci <* configuration_item |
       NOT (SIZEOF (QUERY (ccda <* cc_design_approval |
       ci IN ccda.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

configuration_item: identifies the set of all instances of **configuration_item** entities.

cc_design_approval: identifies the set of all instances of **cc_design_approval** entities.

Formal propositions:

WR1: For each instance of **configuration_item**, there shall be exactly one instance of **cc_design_approval** that contains the instance of **configuration_item** in its set of **items**.

5.2.5.68 coordinated_assembly_and_shape

The **coordinated_assembly_and_shape** rule specifies that the relationship between two **product_definition** entities which are an assembly and a component in a **next_assembly_usage_occurrence** and the relationship between two **shape_representation** entities which contain the representation of the shapes of the assembly and the component in a **shape_representation_relationship** must be explicitly

related through the **context_dependent_shape_representation**. This rule calls the function **assembly_-shape_is_defined** which returns true if the assembly relationship and shape relationship are explicitly related.

EXPRESS specification:

```
*)
RULE coordinated_assembly_and_shape FOR
  (next_assembly_usage_occurrence);
WHERE
  WR1: SIZEOF (QUERY (nauo <* next_assembly_usage_occurrence |
       NOT (assembly_shape_is_defined (nauo, 'CONFIG_CONTROL_DESIGN'))))
       = 0;
END_RULE;
(*
```

Argument definitions:

next_assembly_usage_occurrence: identifies the set of all instances of **next_assembly_usage_occurrence** entities.

Formal propositions:

WR1: For each of instance of **next_assembly_usage_occurrence**, the **assembly_shape_is_defined** function shall be true.

5.2.5.69 subtype_mandatory_product_definition_usage

The **subtype_mandatory_product_definition_usage** rule specifies that all **product_definition_usage** entities shall be **assembly_component_usage** entities.

EXPRESS specification:

```
*)
RULE subtype_mandatory_product_definition_usage FOR
  (product_definition_usage);
WHERE
  WR1: SIZEOF (QUERY (pdu <* product_definition_usage |
       NOT ('CONFIG_CONTROL_DESIGN.' +
       'ASSEMBLY_COMPONENT_USAGE' IN TYPEOF (pdu)))) = 0;
END_RULE;
(*
```

Argument definitions:

product_definition_usage: identifies the set of all instances of **product_definition_usage** entities to be constrained.

Formal propositions:

WR1: Each instance of **product_definition_usage** shall be an **assembly_component_usage**.

5.2.5.70 acu_requires_security_classification

The **acu_requires_security_classification** rule specifies that every **assembly_component_usage** shall be referenced by exactly one **cc_design_security_classification**. This rule enforces the requirement for every **product_definition** which is the **related_product_definition** in a **assembly_component_usage** to have a security classification within the context of the **assembly_component_usage**.

EXPRESS specification:

```
*)
RULE acu_requires_security_classification FOR
  (assembly_component_usage,
   cc_design_security_classification);
WHERE
  WR1: SIZEOF (QUERY (acu <* assembly_component_usage |
       NOT (SIZEOF (QUERY (ccdsc <* cc_design_security_classification |
       acu IN ccdsc.items )) = 1 ))) = 0;
END_RULE;
(*
```

Argument definitions:

assembly_component_usage: identifies the set of all instances of **assembly_component_usage** entities to be constrained.

cc_design_security_classification: identifies the set of all instances of **cc_design_security_classification** entities to be constrained.

Formal propositions:

WR1: For each instance of **assembly_component_usage** there shall be exactly one instance of **cc_design_-security_classification** that contains the instance of **assembly_component_usage** in its set of **items**.

5.2.5.71 geometric_representation_item_3d

The **geometric_representation_item_3d** rule specifies that every **geometric_representation_item** must be founded within a **geometric_representation_context** that has a dimensionality of 3 except for use in defining the **pcurve** entity. This rule constrains all geometry to be three dimensional. The **pcurve** entity is an exception because it must be founded in a 2 dimensional context which is the parameter space of a surface.

EXPRESS specification:

```
*)
RULE geometric_representation_item_3d FOR
  (geometric_representation_item);
WHERE
  WR1: SIZEOF (QUERY (gri <* geometric_representation_item |
       NOT ((dimension_of (gri) = 3) OR
       (SIZEOF (QUERY (ur <* using_representations(gri) |
       'CONFIG_CONTROL_DESIGN.DEFINITIONAL_REPRESENTATION'
       IN TYPEOF(ur))) > 0)))) = 0;
END_RULE;
(*
```

Argument definitions:

geometric_representation_item: identifies the set of all instances of **geometric_representation_item** entities to be constrained.

Formal propositions:

WR1: For each instance of **geometric_representation_item**, the value of **dim** shall be three or the **geometric_representation_item** shall be used as an item in a **definitional_representation**.

> NOTE - The use of the **definitional_representation** is to define points or curves in the parametric space of a surface for use by the **pcurve**.

5.2.5.72 dependent_instantiable_parametric_representation_context

The **dependent_instantiable_parametric_representation_context** rule specifies that all instances of **parametric_representation_context** are dependent on the usage to define another entity.

EXPRESS specification:

```
*)
RULE dependent_instantiable_parametric_representation_context FOR
  (parametric_representation_context);
WHERE
  WR1: SIZEOF (QUERY (prc <* parametric_representation_context |
       NOT (SIZEOF (USEDIN (prc, '')) >= 1))) = 0;
END_RULE;
(*
```

Argument definition:

parametric_representation_context: identifies the set of all instances of **parametric_representation_-context**.

Formal proposition:

WR1: For each instance of **parametric_representation_context**, there shall be a reference to the **parametric_representation_context** instance from an attribute of another entity.

5.2.5.73 subtype_mandatory_shape_representation

The **subtype_mandatory_shape_representation** rule requires all **shape_representation** entities to be either **geometrically_bounded_wireframe_shape_representation**, **geometrically_bounded_surface_-shape_representation**, **edge_based_wireframe_shape_representation**, **shell_based_wireframe_shape_-representation**, **manifold_surface_shape_representation**, **faceted_brep_shape_representation** or **advanced_brep_shape_representation** or contain only **axis2_placement_3d** entities in their respective sets of **items** or be the representation of a **shape_aspect** or a relationship between two **shape_aspect**s. The rule defines the constraint that establishes the different types of representations allowed for shapes in this part of ISO 10303.

EXPRESS specification:

```
*)
RULE subtype_mandatory_shape_representation FOR
  (shape_representation);
```

```
WHERE
  WR1: SIZEOF (QUERY (sr <* shape_representation |
       NOT ((SIZEOF (['CONFIG_CONTROL_DESIGN.' +
       'ADVANCED_BREP_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.FACETED_BREP_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.MANIFOLD_SURFACE_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.' +
       'GEOMETRICALLY_BOUNDED_SURFACE_SHAPE_REPRESENTATION',
       'CONFIG_CONTROL_DESIGN.' +
       'GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION'] *
       TYPEOF (sr)) = 1) OR
       (SIZEOF (QUERY (it <* sr\representation.items |
       NOT ('CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D' IN TYPEOF (it))))
       = 0) OR
       (SIZEOF (QUERY (sdr <* QUERY (pdr <* USEDIN (sr,
       'CONFIG_CONTROL_DESIGN.PROPERTY_DEFINITION_REPRESENTATION.' +
       'USED_REPRESENTATION') |
       'CONFIG_CONTROL_DESIGN.SHAPE_DEFINITION_REPRESENTATION' IN
       TYPEOF (pdr)) |
       NOT (SIZEOF (['CONFIG_CONTROL_DESIGN.SHAPE_ASPECT',
       'CONFIG_CONTROL_DESIGN.SHAPE_ASPECT_RELATIONSHIP'] * TYPEOF
       (sdr.definition.definition)) = 1))) = 0)))) = 0;
END_RULE;
(*
```

Argument definitions:

shape_representation: identifies the set of all instances of **shape_representation** entities to be constrained.

Formal propositions:

WR1: Each instance of **shape_representation** shall be either a **geometrically_bounded_wireframe_-representation**, **geometrically_bounded_surface_representation**, **edge_based_wireframe_-representation**, **shell_based_wireframe_representation**, **manifold_surface_with_topology_-representation**, **faceted_brep_representation** or **advanced_brep_representation** or contain only **axis2_-placement_3d** entities in its set of **items** or be the representation of a **shape_aspect** or **shape_aspect_-relationship**.

5.2.5.74 subtype_mandatory_representation

The **subtype_mandatory_representation** rule requires all **representation** entities to be **shape_-representation**s.

EXPRESS specification:

```
*)
RULE subtype_mandatory_representation FOR (representation);
WHERE
  WR1: SIZEOF (QUERY (rep <* representation |
       NOT ('CONFIG_CONTROL_DESIGN.SHAPE_REPRESENTATION' IN
       TYPEOF (rep)))) = 0;
END_RULE;
(*
```

Argument definitions:

representation: identifies the set of all instances of **representation** entities to be constrained.

Formal propositions:

WR1: Each instance of **representation** shall be a **shape_representation**.

5.2.5.75 subtype_mandatory_representation_context

The **subtype_mandatory_representation_context** rule requires all **representation_context** entities to be **geometric_representation_context**s.

EXPRESS specification:

```
*)
RULE subtype_mandatory_representation_context FOR (representation_context);
WHERE
  WR1: SIZEOF (QUERY (rep_cntxt <* representation_context |
       NOT ('CONFIG_CONTROL_DESIGN.GEOMETRIC_REPRESENTATION_CONTEXT' IN
       TYPEOF (rep_cntxt)))) = 0;
END_RULE;
(*
```

Argument definitions:

representation_context: identifies the set of all instances of **representation_context** entities to be constrained.

Formal propositions:

WR1: Each instance of **representation_context** shall be a **geometric_representation_context**.

5.2.5.76 no_shape_for_make_from

The **no_shape_for_make_from** rule ensures that **product_definition_relationship** entities that are **design_make_from_relationship** entities do not have shape defined for them. This rule works by disallowing the use of a **design_make_from_relationship** entity in a **product_definition_shape** entity which establishes the shape of a part or the relative shape of a constituent part in an assembly.

EXPRESS specification:

```
*)
RULE no_shape_for_make_from FOR
  (design_make_from_relationship);
WHERE
  WR1: SIZEOF (QUERY (dmfr <* design_make_from_relationship |
       NOT (SIZEOF (QUERY (pd <* USEDIN (dmfr, 'CONFIG_CONTROL_DESIGN.' +
       'PROPERTY_DEFINITION.DEFINITION') |
       'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_SHAPE' IN TYPEOF (pd))) =
       0))) = 0;
END_RULE;
(*
```

Argument definitions:

design_make_from_relationship: identifies all instances of the **design_make_from_relationship** entity that are to be constrained.

Formal propositions:

WR1: No **design_make_from_relationship** entities shall be referenced by the **definition** attribute of a **property_definition** that is a **product_definition_shape**.

5.2.5.77 no_shape_for_supplied_part

The **no_shape_for_supplied_part** rule ensures that **product_definition_relationship** entities that are **supplied_part_relationship** entities do not have shape defined for them. This rule works by disallowing the use of a **supplied_part_relationship** entity in a **product_definition_shape** entity which establishes the shape of a part or the relative shape of a constituent part in an assembly.

EXPRESS specification:

```
*)
RULE no_shape_for_supplied_part FOR
  (supplied_part_relationship);
WHERE
  WR1: SIZEOF (QUERY (spr <* supplied_part_relationship |
       NOT (SIZEOF (QUERY (pd <* USEDIN (spr, 'CONFIG_CONTROL_DESIGN.' +
       'PROPERTY_DEFINITION.DEFINITION') |
       'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_SHAPE' IN TYPEOF (pd))) =
       0))) = 0;
END_RULE;
(*
```

Argument definitions:

supplied_part_relationship: identifies all instances of the **supplied_part_relationship** entity that are to be constrained.

Formal propositions:

WR1: No **supplied_part_relationship** entities shall be referenced by the **definition** attribute of a **property_definition** that is a **product_definition_shape**.

5.2.5.78 approval_date_time_constraints

The **approval_date_time_constraints** specifies that each instance of **approval_date_time** shall only reference a **date_and_time** instance. This rule enforces the requirement that dates be specified with a time.

EXPRESS specification:

```
*)
RULE approval_date_time_constraints FOR (approval_date_time);
WHERE
  WR1: SIZEOF (QUERY (adt <* approval_date_time |
```

```
       NOT (SIZEOF (TYPEOF(adt.date_time) *
       ['CONFIG_CONTROL_DESIGN.DATE_AND_TIME']) = 1)))=0;
END_RULE;
(*
```

Argument definitions:

approval_date_time: identifies the set of all instances of **approval_date_time** entities.

Formal propositions:

WR1: For each instance of **approval_date_time**, the **date_time** attribute shall reference an instance of **date_and_time**.

5.2.5.79 approval_person_organization_constraints

The **approval_person_organization_constraints** specifies that each instance of **approval_person_-organization** shall only reference a **person_and_organization** instance. This rule enforces the requirement that people be specified within an organization.

EXPRESS specification:

```
*)
RULE approval_person_organization_constraints FOR
  (approval_person_organization);
WHERE
  WR1: SIZEOF (QUERY (apo <* approval_person_organization |
       NOT (SIZEOF (TYPEOF(apo.person_organization) *
       ['CONFIG_CONTROL_DESIGN.PERSON_AND_ORGANIZATION']) = 1)))=0;
END_RULE;
(*
```

Argument definitions:

approval_person_organization: identifies the set of all instances of **approval_person_organization** entities.

Formal propositions:

WR1: For each instance of **approval_person_organization**, the **person_organization** attribute shall reference an instance of **person_and_organization**.

5.2.6 Configuration controlled design functions

5.2.6.1 unique_version_change_order

The **unique_version_change_order** boolean function accepts an **action_execution** as input and returns true if the **ordered_action** invoked by the **action_execution** incorporates **requested_action** entities which reference different **product_definition_formation** entities and those **product_definition_formation** entities reference different **product** entities. This function will return false if the **requested_action** entities reference **product_definition_formation** entities that are incorporated by an **action_execution** and those **product_-definition_formation** entities reference the same **product**. This function will be true if a change changes multiple versions of a part, and those versions are versions of different parts. A single change may not change different versions of a single part.

EXPRESS specification:

```
*)
FUNCTION unique_version_change_order (c : action) : BOOLEAN;
  LOCAL
    ords       : action_directive := c\directed_action.directive;
    assign     : SET OF change_request := [];
    versions   : SET OF product_definition_formation := [];
  END_LOCAL;

  -- build the set of change_requests that are the assigned
  -- versioned_action_requests incorporated by the action_directive

  REPEAT i := 1 TO SIZEOF(ords.requests);
  assign := assign + QUERY (ara <* bag_to_set (USEDIN (ords.requests[i],
            'CONFIG_CONTROL_DESIGN.ACTION_REQUEST_ASSIGNMENT.' +
            'ASSIGNED_ACTION_REQUEST')) |
            'CONFIG_CONTROL_DESIGN.CHANGE_REQUEST' IN TYPEOF (ara));
  END_REPEAT;

  -- gather the product_definition_formations that are referenced by the
  -- change_requests

  REPEAT k := 1 TO SIZEOF(assign);
    versions := versions + assign[k].items;
  END_REPEAT;

  -- check that no product_definition_formation reference the same
  -- instance of product
```

```
  RETURN (SIZEOF (QUERY (vers <* versions |
          NOT (SIZEOF (QUERY (other_vers <* versions - vers |
          vers.of_product :=: other_vers.of_product)) = 0))) = 0);
END_FUNCTION;
(*
```

Argument definitions:

c: (input) identifies the **directed_action** to be checked.

5.2.6.2 cc_design_person_and_organization_correlation

The **cc_design_person_and_organization_correlation** boolean function returns true if the **name** attribute value of the **person_organization_role** entity is coordinated with the type of entity selected in the **items** of a **cc_design_person_and_organization_assignment** entity.

The function states the following:

— a person and organization assigned to a **change_request** or **start_request** may have the role of "request_recipient";

— a person and organization assigned to a **change_request**, **start_request**, **change**, or **start_work** may have the role of "initiator";

— a person and organization assigned to a **product_definition_formation** or a **product_definition** may have the role of "creator";

— a person and organization assigned to a **product_definition_formation** may have the role of "part_supplier";

— a person and organization assigned to a **product_definition_formation** may have the role of "design_supplier";

— a person and organization assigned to a **product** may have the role of "design_owner";

— a person and organization assigned to a **configuration_item** may have the role of "configuration_manager";

— a person and organization assigned to a **contract** may have the role of "contractor";

— a person and organization assigned to a **security_classification** may have the role of "classification_officer".

EXPRESS specification:

```
*)
FUNCTION cc_design_person_and_organization_correlation
  (e : cc_design_person_and_organization_assignment ) : BOOLEAN;
  LOCAL
    po_role : STRING;
  END_LOCAL;
    po_role := e\person_and_organization_assignment.role.name;
  CASE po_role OF
    'request_recipient'      : IF SIZEOF (e.items) <>
                                 SIZEOF (QUERY (x <* e.items |
                                 SIZEOF(['CONFIG_CONTROL_DESIGN.' +
                                 'CHANGE_REQUEST',
                                 'CONFIG_CONTROL_DESIGN.' +
                                 'START_REQUEST'] *
                                 TYPEOF (x)) = 1))
                                 THEN RETURN(FALSE);
                               END_IF;
    'initiator'              : IF SIZEOF (e.items) <>
                                 SIZEOF (QUERY (x <* e.items |
                                 SIZEOF(['CONFIG_CONTROL_DESIGN.' +
                                 'CHANGE_REQUEST',
                                 'CONFIG_CONTROL_DESIGN.' +
                                 'START_REQUEST',
                                 'CONFIG_CONTROL_DESIGN.' +
                                 'START_WORK',
                                 'CONFIG_CONTROL_DESIGN.' +
                                 'CHANGE'] *
                                 TYPEOF (x)) = 1))
                                 THEN RETURN(FALSE);
                               END_IF;
    'creator'                : IF SIZEOF (e.items) <>
                                 SIZEOF (QUERY (x <* e.items |
                                 SIZEOF (['CONFIG_CONTROL_DESIGN.' +
                                 'PRODUCT_DEFINITION_FORMATION',
                                 'CONFIG_CONTROL_DESIGN.' +
                                 'PRODUCT_DEFINITION'] *
                                 TYPEOF (x)) = 1))
                                 THEN RETURN (FALSE);
                               END_IF;
    'part_supplier'          : IF SIZEOF (e.items) <>
```

```
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'PRODUCT_DEFINITION_FORMATION'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
    'design_supplier'        : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'PRODUCT_DEFINITION_FORMATION'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
    'design_owner'           : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.PRODUCT'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
    'configuration_manager'  : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'CONFIGURATION_ITEM'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
    'contractor'             : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.CONTRACT'
                                  IN TYPEOF (x))) THEN RETURN(FALSE);
                                END_IF;
    'classification_officer' : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'SECURITY_CLASSIFICATION'
                                  IN TYPEOF (x))) THEN RETURN(FALSE);
                                END_IF;
    OTHERWISE : RETURN(TRUE);
  END_CASE;
  RETURN (TRUE);
END_FUNCTION;
(*
```

Argument definitions:

e: (input) identifies the **cc_design_person_and_organization_assignment** to be checked.

5.2.6.3 cc_design_date_time_correlation

The **cc_design_date_time_correlation** boolean function returns true if the **name** attribute value of the **date_time_role** entity is coordinated with the type of entity selected in the **items** of the **cc_design_date_-and_time_assignment** entity.

The function states the following:

— a date and time assigned to a **product_definition** may have the role of "creation_date";

— a date and time assigned to a **change_request** or **start_request** may have the role of "request_date";

— a date and time assigned to a **change** or a **start_work** may have the role of "release_date";

— a date and time assigned to a **change** or a **start_work** may have the role of "start_date";

— a date and time assigned to an **approval_date_time** may have the role of "sign_off_date";

— a date and time assigned to a **contract** may have the role of "contract_date";

— a date and time assigned to a **certification** may have the role of "certification_date";

— a date and time assigned to a **security_classification** may have the role of "classification_date";

— a date and time assigned to a **security_classification** may have the role of "declassification_-date".

EXPRESS specification:

```
*)
FUNCTION cc_design_date_time_correlation
  (e : cc_design_date_and_time_assignment ) : BOOLEAN;
  LOCAL
    dt_role : STRING;
  END_LOCAL;
    dt_role := e\date_and_time_assignment.role.name;
```

```
  CASE dt_role OF
    'creation_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'request_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE_REQUEST',
                                'CONFIG_CONTROL_DESIGN.START_REQUEST'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'release_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE',
                                'CONFIG_CONTROL_DESIGN.START_WORK'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'start_date'          : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE',
                                'CONFIG_CONTROL_DESIGN.START_WORK'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'sign_off_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'APPROVAL_PERSON_ORGANIZATION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'contract_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.CONTRACT'
```

```
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'certification_date'    : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.CERTIFICATION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'classification_date'   : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'SECURITY_CLASSIFICATION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'declassification_date' : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'SECURITY_CLASSIFICATION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    OTHERWISE : RETURN(TRUE);
  END_CASE;
  RETURN (TRUE);
END_FUNCTION;
(*
```

Argument definitions:

e: (input) identifies the **cc_design_date_and_time_assignment** to be checked.

5.2.6.4 assembly_shape_is_defined

The **assembly_shape_is_defined** function accepts a **next_assembly_usage_occurrence** as input and returns a boolean result. The function will return true if the shape is defined for the **product_definition** which is the **related_product_definition** and the shape is defined for the **product_definition** which is the **relating_-product_definition** in the **next_assembly_usage_occurrence**, and the two shapes are related through a **shape_representation_relationship**, and the two relationships are related through **context_dependent_-shape_representation**. The function will also return true if the shape of the **related_product_definition** or the **relating_product_definition** is not defined.

The only time this function will return false is when the shapes are defined for the **related_product_-definition** and the **relating_product_definition** and related through **shape_representation_relationship**, but the **next_assembly_usage_occurrence** and the **shape_representation_relationship** are not explicitly related through **context_dependent_shape_representation**.

EXPRESS specification:

```
*)
FUNCTION assembly_shape_is_defined(
             assy: next_assembly_usage_occurrence;
             schma: STRING
    ): BOOLEAN;

  LOCAL
    srr_set   : SET OF shape_representation_relationship := [];
    i         : INTEGER;
    j         : INTEGER;
    sdr_set   : SET OF shape_definition_representation := [];
    pr1_set   : SET OF property_definition := [];
    pdrel_set : SET OF product_definition_relationship := [];
    pr2_set   : SET OF property_definition := [];
  END_LOCAL;
  pr1_set := bag_to_set(USEDIN(assy.related_product_definition,schma +
      '.PROPERTY_DEFINITION.DEFINITION'));
  REPEAT i := 1 TO HIINDEX(pr1_set) BY 1;
    sdr_set := sdr_set + QUERY ( pdr <* USEDIN(pr1_set[i],schma +
        '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION') | ((schma +
        '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(pdr)) );
  END_REPEAT;
  pdrel_set := bag_to_set(USEDIN(assy.related_product_definition,schma +
       '.PRODUCT_DEFINITION_RELATIONSHIP.' +
       'RELATED_PRODUCT_DEFINITION'));
  REPEAT j := 1 TO HIINDEX(pdrel_set) BY 1;
    pr2_set := pr2_set + USEDIN(pdrel_set[j],schma +
        '.PROPERTY_DEFINITION.DEFINITION');
  END_REPEAT;
  REPEAT i := 1 TO HIINDEX(pr2_set) BY 1;
    sdr_set := sdr_set + QUERY ( pdr <* USEDIN(pr2_set[i],schma +
        '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION') | ((schma +
        '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(pdr)) );
  END_REPEAT;
  IF SIZEOF(sdr_set) > 0 THEN
```

```
    REPEAT i := 1 TO HIINDEX(sdr_set) BY 1;
      srr_set := QUERY ( rr <* bag_to_set(USEDIN(sdr_set[i]\
          property_definition_representation.used_representation,schma +
          '.REPRESENTATION_RELATIONSHIP.REP_2')) | ((schma +
          '.SHAPE_REPRESENTATION_RELATIONSHIP') IN TYPEOF(rr)) );
      IF SIZEOF(srr_set) > 0 THEN
        REPEAT j := 1 TO HIINDEX(srr_set) BY 1;
          IF SIZEOF(QUERY ( pdr <* bag_to_set(USEDIN(srr_set[j]\
              representation_relationship.rep_1,schma +
              '.PROPERTY_DEFINITION_REPRESENTATION.USED_REPRESENTATION'))
               | ((schma + '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(
              pdr)) ) * QUERY ( pdr <* bag_to_set(USEDIN(assy.
              relating_product_definition,schma +
              '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION')) | ((
              schma + '.SHAPE_DEFINITION_REPRESENTATION')
              IN TYPEOF(pdr)) )) >= 1 THEN
            IF SIZEOF(QUERY ( cdsr <* USEDIN(srr_set[j],schma +
                '.CONTEXT_DEPENDENT_SHAPE_REPRESENTATION.' +
                'REPRESENTATION_RELATION') | (NOT (cdsr\
                context_dependent_shape_representation.
                represented_product_relation\property_definition.
                definition :=: assy)) )) > 0 THEN RETURN(FALSE);
            END_IF;
          END_IF;
        END_REPEAT;
      END_IF;
    END_REPEAT;
  END_IF;
  RETURN(TRUE);
END_FUNCTION; -- assembly_shape_is_defined
(*
```

Argument definitions:

assy: (input) identifies the **next_assembly_usage_occurrence** for which the relationships are to be checked.

EXPRESS specification:

```
*)
END_SCHEMA; --config_control_design
(*
```

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

— solid_angle_measure_with_unit;
— solid_angle_unit;
— solid_model;
— specified_higher_usage_occurrence;
— spherical_surface;
— start_request;
— start_work;
— supplied_part_relationship;
— surface;
— surface_of_linear_extrusion;
— surface_of_revolution;
— swept_surface;
— topological_representation_item;
— uncertainty_measure_with_unit;
— uniform_curve;
— uniform_surface;
— vector;
— versioned_action_request;
— vertex;
— vertex_loop;
— vertex_point;
— volume_measure_with_unit;
— volume_unit;
— week_of_year_and_day_date.

Annex A
(normative)

AIM EXPRESS expanded listing

```
*)
SCHEMA config_control_design;

  CONSTANT
  dummy_gri : geometric_representation_item := representation_item('') ||
                 geometric_representation_item();
  dummy_tri : topological_representation_item := representation_item('')
                 || topological_representation_item();
  END_CONSTANT;

  TYPE ahead_or_behind = ENUMERATION OF
    (ahead,
     behind);
  END_TYPE; -- ahead_or_behind

  TYPE approved_item = SELECT
    (product_definition_formation,
     product_definition,
     configuration_effectivity,
     configuration_item,
     security_classification,
     change_request,
     change,
     start_request,
     start_work,
     certification,
     contract);
  END_TYPE; -- approved_item

  TYPE area_measure = REAL;
  END_TYPE; -- area_measure

  TYPE axis2_placement = SELECT
    (axis2_placement_2d,
     axis2_placement_3d);
  END_TYPE; -- axis2_placement
```

```
TYPE b_spline_curve_form = ENUMERATION OF
  (polyline_form,
   circular_arc,
   elliptic_arc,
   parabolic_arc,
   hyperbolic_arc,
   unspecified);
END_TYPE; -- b_spline_curve_form

TYPE b_spline_surface_form = ENUMERATION OF
  (plane_surf,
   cylindrical_surf,
   conical_surf,
   spherical_surf,
   toroidal_surf,
   surf_of_revolution,
   ruled_surf,
   generalised_cone,
   quadric_surf,
   surf_of_linear_extrusion,
   unspecified);
END_TYPE; -- b_spline_surface_form

TYPE boolean_operand = SELECT
  (solid_model);
END_TYPE; -- boolean_operand

TYPE certified_item = SELECT
  (supplied_part_relationship);
END_TYPE; -- certified_item

TYPE change_request_item = SELECT
  (product_definition_formation);
END_TYPE; -- change_request_item

TYPE characterized_definition = SELECT
  (characterized_product_definition,
   shape_definition);
END_TYPE; -- characterized_definition

TYPE characterized_product_definition = SELECT
  (product_definition,
```

```
    product_definition_relationship);
END_TYPE; -- characterized_product_definition

TYPE classified_item = SELECT
  (product_definition_formation,
   assembly_component_usage);
END_TYPE; -- classified_item

TYPE context_dependent_measure = REAL;
END_TYPE; -- context_dependent_measure

TYPE contracted_item = SELECT
  (product_definition_formation);
END_TYPE; -- contracted_item

TYPE count_measure = NUMBER;
END_TYPE; -- count_measure

TYPE curve_on_surface = SELECT
  (pcurve,
   surface_curve,
   composite_curve_on_surface);
END_TYPE; -- curve_on_surface

TYPE date_time_item = SELECT
  (product_definition,
   change_request,
   start_request,
   change,
   start_work,
   approval_person_organization,
   contract,
   security_classification,
   certification);
END_TYPE; -- date_time_item

TYPE date_time_select = SELECT
  (date,
   local_time,
   date_and_time);
END_TYPE; -- date_time_select
```

```
TYPE day_in_month_number = INTEGER;
END_TYPE; -- day_in_month_number

TYPE day_in_week_number = INTEGER;
WHERE
  wr1: ((1 <= SELF) AND (SELF <= 7));
END_TYPE; -- day_in_week_number

TYPE day_in_year_number = INTEGER;
END_TYPE; -- day_in_year_number

TYPE descriptive_measure = STRING;
END_TYPE; -- descriptive_measure

TYPE dimension_count = INTEGER;
WHERE
  wr1: (SELF > 0);
END_TYPE; -- dimension_count

TYPE founded_item_select = SELECT
  (founded_item,
   representation_item);
END_TYPE; -- founded_item_select

TYPE geometric_set_select = SELECT
  (point,
   curve,
   surface);
END_TYPE; -- geometric_set_select

TYPE hour_in_day = INTEGER;
WHERE
  wr1: ((0 <= SELF) AND (SELF < 24));
END_TYPE; -- hour_in_day

TYPE identifier = STRING;
END_TYPE; -- identifier

TYPE knot_type = ENUMERATION OF
  (uniform_knots,
   unspecified,
   quasi_uniform_knots,
```

```
   piecewise_bezier_knots);
END_TYPE; -- knot_type

TYPE label = STRING;
END_TYPE; -- label

TYPE length_measure = REAL;
END_TYPE; -- length_measure

TYPE list_of_reversible_topology_item = LIST [0:?] OF
            reversible_topology_item;
END_TYPE; -- list_of_reversible_topology_item

TYPE mass_measure = REAL;
END_TYPE; -- mass_measure

TYPE measure_value = SELECT
  (length_measure,
   mass_measure,
   plane_angle_measure,
   solid_angle_measure,
   area_measure,
   volume_measure,
   parameter_value,
   context_dependent_measure,
   descriptive_measure,
   positive_length_measure,
   positive_plane_angle_measure,
   count_measure);
END_TYPE; -- measure_value

TYPE minute_in_hour = INTEGER;
WHERE
  wr1: ((0 <= SELF) AND (SELF <= 59));
END_TYPE; -- minute_in_hour

TYPE month_in_year_number = INTEGER;
WHERE
  wr1: ((1 <= SELF) AND (SELF <= 12));
END_TYPE; -- month_in_year_number
```

```
TYPE parameter_value = REAL;
END_TYPE; -- parameter_value

TYPE pcurve_or_surface = SELECT
  (pcurve,
   surface);
END_TYPE; -- pcurve_or_surface

TYPE person_organization_item = SELECT
  (change,
   start_work,
   change_request,
   start_request,
   configuration_item,
   product,
   product_definition_formation,
   product_definition,
   contract,
   security_classification);
END_TYPE; -- person_organization_item

TYPE person_organization_select = SELECT
  (person,
   organization,
   person_and_organization);
END_TYPE; -- person_organization_select

TYPE plane_angle_measure = REAL;
END_TYPE; -- plane_angle_measure

TYPE positive_length_measure = length_measure;
WHERE
  wr1: (SELF > 0);
END_TYPE; -- positive_length_measure

TYPE positive_plane_angle_measure = plane_angle_measure;
WHERE
  wr1: (SELF > 0);
END_TYPE; -- positive_plane_angle_measure

TYPE preferred_surface_curve_representation = ENUMERATION OF
  (curve_3d,
```

```
    pcurve_s1,
    pcurve_s2);
END_TYPE; -- preferred_surface_curve_representation

TYPE reversible_topology = SELECT
  (reversible_topology_item,
   list_of_reversible_topology_item,
   set_of_reversible_topology_item);
END_TYPE; -- reversible_topology

TYPE reversible_topology_item = SELECT
  (edge,
   path,
   face,
   face_bound,
   closed_shell,
   open_shell);
END_TYPE; -- reversible_topology_item

TYPE second_in_minute = REAL;
WHERE
  wr1: ((0 <= SELF) AND (SELF < 60));
END_TYPE; -- second_in_minute

TYPE set_of_reversible_topology_item = SET [0:?] OF
            reversible_topology_item;
END_TYPE; -- set_of_reversible_topology_item

TYPE shape_definition = SELECT
  (product_definition_shape,
   shape_aspect,
   shape_aspect_relationship);
END_TYPE; -- shape_definition

TYPE shell = SELECT
  (vertex_shell,
   wire_shell,
   open_shell,
   closed_shell);
END_TYPE; -- shell
```

```
TYPE si_prefix = ENUMERATION OF
  (exa,
   peta,
   tera,
   giga,
   mega,
   kilo,
   hecto,
   deca,
   deci,
   centi,
   milli,
   micro,
   nano,
   pico,
   femto,
   atto);
END_TYPE; -- si_prefix

TYPE si_unit_name = ENUMERATION OF
  (metre,
   gram,
   second,
   ampere,
   kelvin,
   mole,
   candela,
   radian,
   steradian,
   hertz,
   newton,
   pascal,
   joule,
   watt,
   coulomb,
   volt,
   farad,
   ohm,
   siemens,
   weber,
   tesla,
   henry,
```

```
    degree_celsius,
    lumen,
    lux,
    becquerel,
    gray,
    sievert);
END_TYPE; -- si_unit_name

TYPE solid_angle_measure = REAL;
END_TYPE; -- solid_angle_measure

TYPE source = ENUMERATION OF
  (made,
   bought,
   not_known);
END_TYPE; -- source

TYPE specified_item = SELECT
  (product_definition,
   shape_aspect);
END_TYPE; -- specified_item

TYPE start_request_item = SELECT
  (product_definition_formation);
END_TYPE; -- start_request_item

TYPE supported_item = SELECT
  (action_directive,
   action,
   action_method);
END_TYPE; -- supported_item

TYPE surface_model = SELECT
  (shell_based_surface_model);
END_TYPE; -- surface_model

TYPE text = STRING;
END_TYPE; -- text

TYPE transformation = SELECT
  (item_defined_transformation,
```

```
   functionally_defined_transformation);
END_TYPE; -- transformation

TYPE transition_code = ENUMERATION OF
  (discontinuous,
   continuous,
   cont_same_gradient,
   cont_same_gradient_same_curvature);
END_TYPE; -- transition_code

TYPE trimming_preference = ENUMERATION OF
  (cartesian,
   parameter,
   unspecified);
END_TYPE; -- trimming_preference

TYPE trimming_select = SELECT
  (cartesian_point,
   parameter_value);
END_TYPE; -- trimming_select

TYPE unit = SELECT
  (named_unit);
END_TYPE; -- unit

TYPE vector_or_direction = SELECT
  (vector,
   direction);
END_TYPE; -- vector_or_direction

TYPE volume_measure = REAL;
END_TYPE; -- volume_measure

TYPE week_in_year_number = INTEGER;
WHERE
  wr1: ((1 <= SELF) AND (SELF <= 53));
END_TYPE; -- week_in_year_number

TYPE wireframe_model = SELECT
  (shell_based_wireframe_model,
   edge_based_wireframe_model);
END_TYPE; -- wireframe_model
```

```
TYPE work_item = SELECT
  (product_definition_formation);
END_TYPE; -- work_item

TYPE year_number = INTEGER;
END_TYPE; -- year_number

ENTITY action;
    name          : label;
    description   : text;
    chosen_method : action_method;
END_ENTITY; -- action

ENTITY action_assignment
  ABSTRACT SUPERTYPE;
    assigned_action : action;
END_ENTITY; -- action_assignment

ENTITY action_directive;
    name        : label;
    description : text;
    analysis    : text;
    comment     : text;
    requests    : SET [1:?] OF versioned_action_request;
END_ENTITY; -- action_directive

ENTITY action_method;
    name        : label;
    description : text;
    consequence : text;
    purpose     : text;
END_ENTITY; -- action_method

ENTITY action_request_assignment
  ABSTRACT SUPERTYPE;
    assigned_action_request : versioned_action_request;
END_ENTITY; -- action_request_assignment

ENTITY action_request_solution;
    method  : action_method;
    request : versioned_action_request;
END_ENTITY; -- action_request_solution
```

```
ENTITY action_request_status;
    status           : label;
    assigned_request : versioned_action_request;
END_ENTITY; -- action_request_status

ENTITY action_status;
    status          : label;
    assigned_action : executed_action;
END_ENTITY; -- action_status

ENTITY address;
    internal_location       : OPTIONAL label;
    street_number           : OPTIONAL label;
    street                  : OPTIONAL label;
    postal_box              : OPTIONAL label;
    town                    : OPTIONAL label;
    region                  : OPTIONAL label;
    postal_code             : OPTIONAL label;
    country                 : OPTIONAL label;
    facsimile_number        : OPTIONAL label;
    telephone_number        : OPTIONAL label;
    electronic_mail_address : OPTIONAL label;
    telex_number            : OPTIONAL label;
  WHERE
    wr1: (EXISTS(internal_location) OR EXISTS(street_number) OR EXISTS(
             street) OR EXISTS(postal_box) OR EXISTS(town) OR EXISTS(
             region) OR EXISTS(postal_code) OR EXISTS(country) OR EXISTS(
             facsimile_number) OR EXISTS(telephone_number) OR EXISTS(
             electronic_mail_address) OR EXISTS(telex_number));
END_ENTITY; -- address

ENTITY advanced_brep_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1: (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF([
             'CONFIG_CONTROL_DESIGN.MANIFOLD_SOLID_BREP',
             'CONFIG_CONTROL_DESIGN.FACETED_BREP',
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
             'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
             1)) )) = 0);
    wr2: (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF([
             'CONFIG_CONTROL_DESIGN.MANIFOLD_SOLID_BREP',
```

```
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) )) >
               0);
    wr3: (SIZEOF(QUERY ( msb <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) )
               | (NOT (SIZEOF(QUERY ( csh <* msb_shells(msb) | (NOT (
              SIZEOF(QUERY ( fcs <* csh\connected_face_set.cfs_faces | (
              NOT ('CONFIG_CONTROL_DESIGN.ADVANCED_FACE'
              IN TYPEOF(fcs))) )) = 0)) )) = 0)) )) = 0);
    wr4: (SIZEOF(QUERY ( msb <* QUERY ( it <* items | (
              'CONFIG_CONTROL_DESIGN.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) )
               | ('CONFIG_CONTROL_DESIGN.ORIENTED_CLOSED_SHELL' IN TYPEOF(
              msb\manifold_solid_brep.outer)) )) = 0);
    wr5: (SIZEOF(QUERY ( brv <* QUERY ( it <* items | (
              'CONFIG_CONTROL_DESIGN.BREP_WITH_VOIDS' IN TYPEOF(it)) ) | (
              NOT (SIZEOF(QUERY ( csh <* brv\brep_with_voids.voids | csh\
              oriented_closed_shell.orientation )) = 0)) )) = 0);
    wr6: (SIZEOF(QUERY ( mi <* QUERY ( it <* items | (
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (NOT
              ('CONFIG_CONTROL_DESIGN.ADVANCED_BREP_SHAPE_REPRESENTATION'
              IN TYPEOF(mi\mapped_item.mapping_source.
              mapped_representation))) )) = 0);
END_ENTITY; -- advanced_brep_shape_representation

ENTITY advanced_face
  SUBTYPE OF (face_surface);
  WHERE
    wr1 : (SIZEOF(['CONFIG_CONTROL_DESIGN.ELEMENTARY_SURFACE',
               'CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE',
               'CONFIG_CONTROL_DESIGN.SWEPT_SURFACE'] * TYPEOF(
               face_geometry)) = 1);
    wr2 : (SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* bounds | (
               'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
                | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds.bound\path.
               edge_list | (NOT ('CONFIG_CONTROL_DESIGN.EDGE_CURVE' IN
               TYPEOF(oe\oriented_edge.edge_element))) )) = 0)) )) = 0);
    wr3 : (SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* bounds | (
               'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
                | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds.bound\path.
               edge_list | (NOT (SIZEOF(['CONFIG_CONTROL_DESIGN.LINE',
               'CONFIG_CONTROL_DESIGN.CONIC',
               'CONFIG_CONTROL_DESIGN.POLYLINE',
               'CONFIG_CONTROL_DESIGN.SURFACE_CURVE',
```

```
          'CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE'] * TYPEOF(oe.
          edge_element\edge_curve.edge_geometry)) = 1)) )) = 0)) )) =
           0);
wr4 : (SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* bounds | (
          'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
           | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds.bound\path.
          edge_list | (NOT (('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN
          TYPEOF(oe\edge.edge_start)) AND (
          'CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN TYPEOF(oe\edge.
          edge_start\vertex_point.vertex_geometry)) AND (
          'CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(oe\edge.
          edge_end)) AND ('CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN
          TYPEOF(oe\edge.edge_end\vertex_point.vertex_geometry)))) ))
          = 0)) )) = 0);
wr5 : (SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* bounds | (
          'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
           | ('CONFIG_CONTROL_DESIGN.ORIENTED_PATH' IN TYPEOF(
          elp_fbnds.bound)) )) = 0);
wr6 : ((NOT ('CONFIG_CONTROL_DESIGN.SWEPT_SURFACE' IN TYPEOF(
          face_geometry))) OR (SIZEOF(['CONFIG_CONTROL_DESIGN.LINE',
          'CONFIG_CONTROL_DESIGN.CONIC',
          'CONFIG_CONTROL_DESIGN.POLYLINE',
          'CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE'] * TYPEOF(
          face_geometry\swept_surface.swept_curve)) = 1));
wr7 : (SIZEOF(QUERY ( vlp_fbnds <* QUERY ( bnds <* bounds | (
          'CONFIG_CONTROL_DESIGN.VERTEX_LOOP'
          IN TYPEOF(bnds.bound)) )
           | (NOT (('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(
          vlp_fbnds\face_bound.bound\vertex_loop.loop_vertex)) AND (
          'CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN TYPEOF(vlp_fbnds
          \face_bound.bound\vertex_loop.loop_vertex\vertex_point.
          vertex_geometry)))) )) = 0);
wr8 : (SIZEOF(QUERY ( bnd <* bounds | (NOT (SIZEOF([
          'CONFIG_CONTROL_DESIGN.EDGE_LOOP',
          'CONFIG_CONTROL_DESIGN.VERTEX_LOOP'] * TYPEOF(bnd.bound)) =
           1)) )) = 0);
wr9 : (SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* bounds | (
          'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
           | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds.bound\path.
          edge_list | (('CONFIG_CONTROL_DESIGN.SURFACE_CURVE' IN
          TYPEOF(oe\oriented_edge.edge_element\edge_curve.
          edge_geometry)) AND (NOT (SIZEOF(QUERY ( sc_ag <* oe.
```

```
                edge_element\edge_curve.edge_geometry\surface_curve.
                associated_geometry | (NOT ('CONFIG_CONTROL_DESIGN.PCURVE'
                IN TYPEOF(sc_ag))) )) = 0))) )) = 0)) )) = 0);
    wr10: (((NOT ('CONFIG_CONTROL_DESIGN.SWEPT_SURFACE' IN TYPEOF(
                face_geometry))) OR (NOT ('CONFIG_CONTROL_DESIGN.POLYLINE'
                IN TYPEOF(face_geometry\swept_surface.swept_curve))) OR (
                SIZEOF(face_geometry\swept_surface.swept_curve\polyline.
                points) >= 3)) AND (SIZEOF(QUERY ( elp_fbnds <*
                QUERY ( bnds <* bounds | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP'
                IN TYPEOF(bnds.bound)) ) | (NOT (SIZEOF(QUERY ( oe <*
                elp_fbnds.bound\path.edge_list | ((
                'CONFIG_CONTROL_DESIGN.POLYLINE' IN TYPEOF(oe\oriented_edge
                .edge_element\edge_curve.edge_geometry)) AND (NOT (SIZEOF(
                oe\oriented_edge.edge_element\edge_curve.edge_geometry\
                polyline.points) >= 3))) )) = 0)) )) = 0));
END_ENTITY; -- advanced_face

ENTITY alternate_product_relationship;
    name       : label;
    definition : text;
    alternate  : product;
    base       : product;
    basis      : text;
  UNIQUE
    ur1 : alternate, base;
  WHERE
    wr1: (alternate :<>: base);
END_ENTITY; -- alternate_product_relationship

ENTITY application_context;
    application : text;
  INVERSE
    context_elements : SET [1:?] OF application_context_element FOR
                           frame_of_reference;
END_ENTITY; -- application_context

ENTITY application_context_element
  SUPERTYPE OF (ONEOF (product_context,product_definition_context,
      product_concept_context));
    name               : label;
    frame_of_reference : application_context;
END_ENTITY; -- application_context_element
```

```
ENTITY application_protocol_definition;
    status                                    : label;
    application_interpreted_model_schema_name : label;
    application_protocol_year                 : year_number;
    application                               : application_context;
END_ENTITY; -- application_protocol_definition

ENTITY approval;
    status : approval_status;
    level  : label;
END_ENTITY; -- approval

ENTITY approval_assignment
  ABSTRACT SUPERTYPE;
    assigned_approval : approval;
END_ENTITY; -- approval_assignment

ENTITY approval_date_time;
    date_time      : date_time_select;
    dated_approval : approval;
END_ENTITY; -- approval_date_time

ENTITY approval_person_organization;
    person_organization : person_organization_select;
    authorized_approval : approval;
    role                : approval_role;
END_ENTITY; -- approval_person_organization

ENTITY approval_relationship;
    name              : label;
    description       : text;
    relating_approval : approval;
    related_approval  : approval;
END_ENTITY; -- approval_relationship

ENTITY approval_role;
    role : label;
END_ENTITY; -- approval_role

ENTITY approval_status;
    name : label;
END_ENTITY; -- approval_status
```

```
ENTITY area_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.AREA_UNIT' IN TYPEOF(SELF\
             measure_with_unit.unit_component));
END_ENTITY; -- area_measure_with_unit

ENTITY area_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 2) AND (SELF\
             named_unit.dimensions.mass_exponent = 0) AND (SELF\
             named_unit.dimensions.time_exponent = 0) AND (SELF\
             named_unit.dimensions.electric_current_exponent = 0) AND (
             SELF\named_unit.dimensions.
             thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
             .dimensions.amount_of_substance_exponent = 0) AND (SELF\
             named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- area_unit

ENTITY assembly_component_usage
  SUPERTYPE OF (ONEOF (next_assembly_usage_occurrence,
      specified_higher_usage_occurrence,promissory_usage_occurrence))
  SUBTYPE OF (product_definition_usage);
    reference_designator : OPTIONAL identifier;
END_ENTITY; -- assembly_component_usage

ENTITY assembly_component_usage_substitute;
    name       : label;
    definition : text;
    base       : assembly_component_usage;
    substitute : assembly_component_usage;
  UNIQUE
    ur1 : base, substitute;
  WHERE
    wr1: (base.relating_product_definition :=: substitute.
             relating_product_definition);
    wr2: (base :<>: substitute);
END_ENTITY; -- assembly_component_usage_substitute

ENTITY axis1_placement
  SUBTYPE OF (placement);
```

```
    axis : OPTIONAL direction;
  DERIVE
    z : direction := NVL(normalise(axis),dummy_gri ||
           direction([0,0,1]));
  WHERE
    wr1: (SELF\geometric_representation_item.dim = 3);
END_ENTITY; -- axis1_placement

ENTITY axis2_placement_2d
  SUBTYPE OF (placement);
    ref_direction : OPTIONAL direction;
  DERIVE
    p : LIST [2:2] OF direction := build_2axes(ref_direction);
  WHERE
    wr1: (SELF\geometric_representation_item.dim = 2);
END_ENTITY; -- axis2_placement_2d

ENTITY axis2_placement_3d
  SUBTYPE OF (placement);
    axis          : OPTIONAL direction;
    ref_direction : OPTIONAL direction;
  DERIVE
    p : LIST [3:3] OF direction := build_axes(axis,ref_direction);
  WHERE
    wr1: (SELF\placement.location.dim = 3);
    wr2: ((NOT EXISTS(axis)) OR (axis.dim = 3));
    wr3: ((NOT EXISTS(ref_direction)) OR (ref_direction.dim = 3));
    wr4: ((NOT EXISTS(axis)) OR (NOT EXISTS(ref_direction)) OR (
             cross_product(axis,ref_direction).magnitude > 0));
END_ENTITY; -- axis2_placement_3d

ENTITY b_spline_curve
  SUPERTYPE OF (ONEOF (uniform_curve,b_spline_curve_with_knots,
      quasi_uniform_curve,bezier_curve) ANDOR rational_b_spline_curve)
  SUBTYPE OF (bounded_curve);
    degree              : INTEGER;
    control_points_list : LIST [2:?] OF cartesian_point;
    curve_form          : b_spline_curve_form;
    closed_curve        : LOGICAL;
    self_intersect      : LOGICAL;
  DERIVE
    upper_index_on_control_points : INTEGER := SIZEOF(
```

```
                                        control_points_list) - 1;
    control_points                : ARRAY [0:
                                        upper_index_on_control_points] OF
                                         cartesian_point := list_to_array(
                                        control_points_list,0,
                                        upper_index_on_control_points);
  WHERE
    wr1: (('CONFIG_CONTROL_DESIGN.UNIFORM_CURVE' IN TYPEOF(SELF)) OR (
             'CONFIG_CONTROL_DESIGN.QUASI_UNIFORM_CURVE' IN TYPEOF(SELF))
             OR ('CONFIG_CONTROL_DESIGN.BEZIER_CURVE' IN TYPEOF(SELF)) OR
              ('CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE_WITH_KNOTS' IN
             TYPEOF(SELF)));
END_ENTITY; -- b_spline_curve

ENTITY b_spline_curve_with_knots
  SUBTYPE OF (b_spline_curve);
    knot_multiplicities : LIST [2:?] OF INTEGER;
    knots               : LIST [2:?] OF parameter_value;
    knot_spec           : knot_type;
  DERIVE
    upper_index_on_knots : INTEGER := SIZEOF(knots);
  WHERE
    wr1: constraints_param_b_spline(degree,upper_index_on_knots,
             upper_index_on_control_points,knot_multiplicities,knots);
    wr2: (SIZEOF(knot_multiplicities) = upper_index_on_knots);
END_ENTITY; -- b_spline_curve_with_knots

ENTITY b_spline_surface
  SUPERTYPE OF (ONEOF (b_spline_surface_with_knots,uniform_surface,
      quasi_uniform_surface,bezier_surface) ANDOR
      rational_b_spline_surface)
  SUBTYPE OF (bounded_surface);
    u_degree            : INTEGER;
    v_degree            : INTEGER;
    control_points_list : LIST [2:?] OF LIST [2:?] OF cartesian_point;
    surface_form        : b_spline_surface_form;
    u_closed            : LOGICAL;
    v_closed            : LOGICAL;
    self_intersect      : LOGICAL;
  DERIVE
    u_upper        : INTEGER := SIZEOF(control_points_list) - 1;
    v_upper        : INTEGER := SIZEOF(control_points_list[1]) - 1;
```

```
    control_points : ARRAY [0:u_upper] OF ARRAY [0:v_upper] OF
                         cartesian_point := make_array_of_array(
                        control_points_list,0,u_upper,0,v_upper);
  WHERE
    wr1: (('CONFIG_CONTROL_DESIGN.UNIFORM_SURFACE' IN TYPEOF(SELF)) OR (
              'CONFIG_CONTROL_DESIGN.QUASI_UNIFORM_SURFACE' IN
              TYPEOF(SELF))
              OR ('CONFIG_CONTROL_DESIGN.BEZIER_SURFACE' IN TYPEOF(SELF))
              OR ('CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE_WITH_KNOTS' IN
              TYPEOF(SELF)));
END_ENTITY; -- b_spline_surface

ENTITY b_spline_surface_with_knots
  SUBTYPE OF (b_spline_surface);
    u_multiplicities : LIST [2:?] OF INTEGER;
    v_multiplicities : LIST [2:?] OF INTEGER;
    u_knots          : LIST [2:?] OF parameter_value;
    v_knots          : LIST [2:?] OF parameter_value;
    knot_spec        : knot_type;
  DERIVE
    knot_u_upper : INTEGER := SIZEOF(u_knots);
    knot_v_upper : INTEGER := SIZEOF(v_knots);
  WHERE
    wr1: constraints_param_b_spline(SELF\b_spline_surface.u_degree,
              knot_u_upper,SELF\b_spline_surface.u_upper,u_multiplicities,
              u_knots);
    wr2: constraints_param_b_spline(SELF\b_spline_surface.v_degree,
              knot_v_upper,SELF\b_spline_surface.v_upper,v_multiplicities,
              v_knots);
    wr3: (SIZEOF(u_multiplicities) = knot_u_upper);
    wr4: (SIZEOF(v_multiplicities) = knot_v_upper);
END_ENTITY; -- b_spline_surface_with_knots

ENTITY bezier_curve
  SUBTYPE OF (b_spline_curve);
END_ENTITY; -- bezier_curve

ENTITY bezier_surface
  SUBTYPE OF (b_spline_surface);
END_ENTITY; -- bezier_surface
```

```
ENTITY boundary_curve
  SUBTYPE OF (composite_curve_on_surface);
  WHERE
    wr1: SELF\composite_curve.closed_curve;
END_ENTITY; -- boundary_curve

ENTITY bounded_curve
  SUPERTYPE OF (ONEOF (polyline,b_spline_curve,trimmed_curve,
      bounded_pcurve,bounded_surface_curve,composite_curve))
  SUBTYPE OF (curve);
END_ENTITY; -- bounded_curve

ENTITY bounded_pcurve
  SUBTYPE OF (pcurve, bounded_curve);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.BOUNDED_CURVE' IN TYPEOF(SELF\pcurve.
             reference_to_curve.items[1]));
END_ENTITY; -- bounded_pcurve

ENTITY bounded_surface
  SUPERTYPE OF (ONEOF (b_spline_surface,rectangular_trimmed_surface,
      curve_bounded_surface,rectangular_composite_surface))
  SUBTYPE OF (surface);
END_ENTITY; -- bounded_surface

ENTITY bounded_surface_curve
  SUBTYPE OF (surface_curve, bounded_curve);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.BOUNDED_CURVE' IN TYPEOF(SELF\
             surface_curve.curve_3d));
END_ENTITY; -- bounded_surface_curve

ENTITY brep_with_voids
  SUBTYPE OF (manifold_solid_brep);
    voids : SET [1:?] OF oriented_closed_shell;
END_ENTITY; -- brep_with_voids

ENTITY calendar_date
  SUBTYPE OF (date);
    day_component   : day_in_month_number;
    month_component : month_in_year_number;
  WHERE
```

```
    wr1: valid_calendar_date(SELF);
END_ENTITY; -- calendar_date

ENTITY cartesian_point
  SUBTYPE OF (point);
    coordinates : LIST [1:3] OF length_measure;
END_ENTITY; -- cartesian_point

ENTITY cartesian_transformation_operator
  SUPERTYPE OF (cartesian_transformation_operator_3d)
  SUBTYPE OF (geometric_representation_item,
      functionally_defined_transformation);
    axis1        : OPTIONAL direction;
    axis2        : OPTIONAL direction;
    local_origin : cartesian_point;
    scale        : OPTIONAL REAL;
  DERIVE
    scl : REAL := NVL(scale,1);
  WHERE
    wr1: (scl > 0);
END_ENTITY; -- cartesian_transformation_operator

ENTITY cartesian_transformation_operator_3d
  SUBTYPE OF (cartesian_transformation_operator);
    axis3 : OPTIONAL direction;
  DERIVE
    u : LIST [3:3] OF direction := base_axis(3,SELF\
           cartesian_transformation_operator.axis1,SELF\
           cartesian_transformation_operator.axis2,axis3);
  WHERE
    wr1: (SELF\geometric_representation_item.dim = 3);
END_ENTITY; -- cartesian_transformation_operator_3d

ENTITY cc_design_approval
  SUBTYPE OF (approval_assignment);
    items : SET [1:?] OF approved_item;
END_ENTITY; -- cc_design_approval

ENTITY cc_design_certification
  SUBTYPE OF (certification_assignment);
    items : SET [1:?] OF certified_item;
END_ENTITY; -- cc_design_certification
```

```
ENTITY cc_design_contract
  SUBTYPE OF (contract_assignment);
    items : SET [1:?] OF contracted_item;
END_ENTITY; -- cc_design_contract

ENTITY cc_design_date_and_time_assignment
  SUBTYPE OF (date_and_time_assignment);
    items : SET [1:?] OF date_time_item;
  WHERE
    wr1: cc_design_date_time_correlation(SELF);
END_ENTITY; -- cc_design_date_and_time_assignment

ENTITY cc_design_person_and_organization_assignment
  SUBTYPE OF (person_and_organization_assignment);
    items : SET [1:?] OF person_organization_item;
  WHERE
    wr1: cc_design_person_and_organization_correlation(SELF);
END_ENTITY; -- cc_design_person_and_organization_assignment

ENTITY cc_design_security_classification
  SUBTYPE OF (security_classification_assignment);
    items : SET [1:?] OF classified_item;
END_ENTITY; -- cc_design_security_classification

ENTITY cc_design_specification_reference
  SUBTYPE OF (document_reference);
    items : SET [1:?] OF specified_item;
END_ENTITY; -- cc_design_specification_reference

ENTITY certification;
    name    : label;
    purpose : text;
    kind    : certification_type;
END_ENTITY; -- certification

ENTITY certification_assignment
  ABSTRACT SUPERTYPE;
    assigned_certification : certification;
END_ENTITY; -- certification_assignment
```

```
ENTITY certification_type;
    description : label;
END_ENTITY; -- certification_type

ENTITY change
  SUBTYPE OF (action_assignment);
    items : SET [1:?] OF work_item;
END_ENTITY; -- change

ENTITY change_request
  SUBTYPE OF (action_request_assignment);
    items : SET [1:?] OF change_request_item;
END_ENTITY; -- change_request

ENTITY circle
  SUBTYPE OF (conic);
    radius : positive_length_measure;
END_ENTITY; -- circle

ENTITY closed_shell
  SUBTYPE OF (connected_face_set);
END_ENTITY; -- closed_shell

ENTITY composite_curve
  SUBTYPE OF (bounded_curve);
    segments       : LIST [1:?] OF composite_curve_segment;
    self_intersect : LOGICAL;
  DERIVE
    n_segments   : INTEGER := SIZEOF(segments);
    closed_curve : LOGICAL := segments[n_segments].transition <>
                       discontinuous;
  WHERE
    wr1: (((NOT closed_curve) AND (SIZEOF(QUERY ( temp <* segments | (
              temp.transition = discontinuous) )) = 1)) OR (closed_curve
              AND (SIZEOF(QUERY ( temp <* segments | (temp.transition =
              discontinuous) )) = 0)));
END_ENTITY; -- composite_curve

ENTITY composite_curve_on_surface
  SUPERTYPE OF (boundary_curve)
  SUBTYPE OF (composite_curve);
  DERIVE
```

```
    basis_surface : SET [0:2] OF surface := get_basis_surface(SELF);
  WHERE
    wr1: (SIZEOF(basis_surface) > 0);
    wr2: constraints_composite_curve_on_surface(SELF);
END_ENTITY; -- composite_curve_on_surface

ENTITY composite_curve_segment
  SUBTYPE OF (founded_item);
    transition   : transition_code;
    same_sense   : BOOLEAN;
    parent_curve : curve;
  INVERSE
    using_curves : BAG [1:?] OF composite_curve FOR segments;
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.BOUNDED_CURVE' IN TYPEOF(parent_curve));
END_ENTITY; -- composite_curve_segment

ENTITY configuration_design;
    configuration : configuration_item;
    design        : product_definition_formation;
  UNIQUE
    ur1 : configuration, design;
END_ENTITY; -- configuration_design

ENTITY configuration_effectivity
  SUBTYPE OF (product_definition_effectivity);
    configuration : configuration_design;
  UNIQUE
    ur1 : configuration, usage, id;
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_USAGE' IN TYPEOF(
             SELF\product_definition_effectivity.usage));
END_ENTITY; -- configuration_effectivity

ENTITY configuration_item;
    id           : identifier;
    name         : label;
    description  : OPTIONAL text;
    item_concept : product_concept;
    purpose      : OPTIONAL label;
```

```
  UNIQUE
    ur1 : id;
END_ENTITY; -- configuration_item

ENTITY conic
  SUPERTYPE OF (ONEOF (circle,ellipse,hyperbola,parabola))
  SUBTYPE OF (curve);
    position : axis2_placement;
END_ENTITY; -- conic

ENTITY conical_surface
  SUBTYPE OF (elementary_surface);
    radius     : length_measure;
    semi_angle : plane_angle_measure;
  WHERE
    wr1: (radius >= 0);
END_ENTITY; -- conical_surface

ENTITY connected_edge_set
  SUBTYPE OF (topological_representation_item);
    ces_edges : SET [1:?] OF edge;
END_ENTITY; -- connected_edge_set

ENTITY connected_face_set
  SUPERTYPE OF (ONEOF (closed_shell,open_shell))
  SUBTYPE OF (topological_representation_item);
    cfs_faces : SET [1:?] OF face;
END_ENTITY; -- connected_face_set

ENTITY context_dependent_shape_representation;
    representation_relation      : shape_representation_relationship;
    represented_product_relation : product_definition_shape;
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_RELATIONSHIP' IN
             TYPEOF(SELF.represented_product_relation.definition));
END_ENTITY; -- context_dependent_shape_representation

ENTITY context_dependent_unit
  SUBTYPE OF (named_unit);
    name : label;
END_ENTITY; -- context_dependent_unit
```

```
ENTITY contract;
    name    : label;
    purpose : text;
    kind    : contract_type;
END_ENTITY; -- contract

ENTITY contract_assignment
  ABSTRACT SUPERTYPE;
    assigned_contract : contract;
END_ENTITY; -- contract_assignment

ENTITY contract_type;
    description : label;
END_ENTITY; -- contract_type

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

ENTITY coordinated_universal_time_offset;
    hour_offset   : hour_in_day;
    minute_offset : OPTIONAL minute_in_hour;
    sense         : ahead_or_behind;
END_ENTITY; -- coordinated_universal_time_offset

ENTITY curve
  SUPERTYPE OF (ONEOF (line,conic,pcurve,surface_curve,offset_curve_3d,
      curve_replica))
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- curve

ENTITY curve_bounded_surface
  SUBTYPE OF (bounded_surface);
    basis_surface  : surface;
    boundaries     : SET [1:?] OF boundary_curve;
    implicit_outer : BOOLEAN;
  WHERE
    wr1: (NOT (implicit_outer AND (
             'CONFIG_CONTROL_DESIGN.OUTER_BOUNDARY_CURVE' IN TYPEOF(
             boundaries))));
```

```
    wr2: ((NOT implicit_outer) OR (
             'CONFIG_CONTROL_DESIGN.BOUNDED_SURFACE' IN TYPEOF(
             basis_surface)));
    wr3: (SIZEOF(QUERY ( temp <* boundaries | (
             'CONFIG_CONTROL_DESIGN.OUTER_BOUNDARY_CURVE'
             IN TYPEOF(temp)) )) <= 1);
    wr4: (SIZEOF(QUERY ( temp <* boundaries | (temp\
             composite_curve_on_surface.basis_surface[1] <> SELF.
             basis_surface) )) = 0);
END_ENTITY; -- curve_bounded_surface

ENTITY curve_replica
  SUBTYPE OF (curve);
    parent_curve   : curve;
    transformation : cartesian_transformation_operator;
  WHERE
    wr1: (transformation.dim = parent_curve.dim);
    wr2: acyclic_curve_replica(SELF,parent_curve);
END_ENTITY; -- curve_replica

ENTITY cylindrical_surface
  SUBTYPE OF (elementary_surface);
    radius : positive_length_measure;
END_ENTITY; -- cylindrical_surface

ENTITY date
  SUPERTYPE OF (ONEOF (calendar_date,ordinal_date,
      week_of_year_and_day_date));
    year_component : year_number;
END_ENTITY; -- date

ENTITY date_and_time;
    date_component : date;
    time_component : local_time;
END_ENTITY; -- date_and_time

ENTITY date_and_time_assignment
  ABSTRACT SUPERTYPE;
    assigned_date_and_time : date_and_time;
    role                   : date_time_role;
END_ENTITY; -- date_and_time_assignment
```

```
ENTITY date_time_role;
    name : label;
END_ENTITY; -- date_time_role

ENTITY dated_effectivity
  SUBTYPE OF (effectivity);
    effectivity_start_date : date_and_time;
    effectivity_end_date   : OPTIONAL date_and_time;
END_ENTITY; -- dated_effectivity

ENTITY definitional_representation
  SUBTYPE OF (representation);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.PARAMETRIC_REPRESENTATION_CONTEXT' IN
             TYPEOF(SELF\representation.context_of_items));
END_ENTITY; -- definitional_representation

ENTITY degenerate_pcurve
  SUBTYPE OF (point);
    basis_surface      : surface;
    reference_to_curve : definitional_representation;
  WHERE
    wr1: (SIZEOF(reference_to_curve\representation.items) = 1);
    wr2: ('CONFIG_CONTROL_DESIGN.CURVE' IN TYPEOF(reference_to_curve\
             representation.items[1]));
    wr3: (reference_to_curve\representation.items[1]\
             geometric_representation_item.dim = 2);
END_ENTITY; -- degenerate_pcurve

ENTITY degenerate_toroidal_surface
  SUBTYPE OF (toroidal_surface);
    select_outer : BOOLEAN;
  WHERE
    wr1: (major_radius < minor_radius);
END_ENTITY; -- degenerate_toroidal_surface

ENTITY design_context
  SUBTYPE OF (product_definition_context);
  WHERE
    wr1: (SELF.life_cycle_stage = 'design');
END_ENTITY; -- design_context
```

```
ENTITY design_make_from_relationship
  SUBTYPE OF (product_definition_relationship);
END_ENTITY; -- design_make_from_relationship

ENTITY dimensional_exponents;
    length_exponent                    : REAL;
    mass_exponent                      : REAL;
    time_exponent                      : REAL;
    electric_current_exponent          : REAL;
    thermodynamic_temperature_exponent : REAL;
    amount_of_substance_exponent       : REAL;
    luminous_intensity_exponent        : REAL;
END_ENTITY; -- dimensional_exponents

ENTITY directed_action
  SUBTYPE OF (executed_action);
    directive : action_directive;
END_ENTITY; -- directed_action

ENTITY direction
  SUBTYPE OF (geometric_representation_item);
    direction_ratios : LIST [2:3] OF REAL;
  WHERE
    wr1: (SIZEOF(QUERY ( tmp <* direction_ratios | (tmp <> 0) )) > 0);
END_ENTITY; -- direction

ENTITY document;
    id          : identifier;
    name        : label;
    description : text;
    kind        : document_type;
  UNIQUE
    ur1 : id;
END_ENTITY; -- document

ENTITY document_reference
  ABSTRACT SUPERTYPE;
    assigned_document : document;
    source            : label;
END_ENTITY; -- document_reference
```

```
ENTITY document_relationship;
    name              : label;
    description       : text;
    relating_document : document;
    related_document  : document;
END_ENTITY; -- document_relationship

ENTITY document_type;
    product_data_type : label;
END_ENTITY; -- document_type

ENTITY document_usage_constraint;
    source                : document;
    subject_element       : label;
    subject_element_value : text;
END_ENTITY; -- document_usage_constraint

ENTITY document_with_class
  SUBTYPE OF (document);
    class : identifier;
END_ENTITY; -- document_with_class

ENTITY edge
  SUPERTYPE OF (ONEOF (edge_curve,oriented_edge))
  SUBTYPE OF (topological_representation_item);
    edge_start : vertex;
    edge_end   : vertex;
END_ENTITY; -- edge

ENTITY edge_based_wireframe_model
  SUBTYPE OF (geometric_representation_item);
    ebwm_boundary : SET [1:?] OF connected_edge_set;
END_ENTITY; -- edge_based_wireframe_model

ENTITY edge_based_wireframe_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1: (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF([
              'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL',
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
              'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
              1)) )) = 0);
```

```
wr2: (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF([
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL',
         'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) ))
         >= 1);
wr3: (SIZEOF(QUERY ( ebwm <* QUERY ( it <* SELF.items | (
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL' IN
         TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( eb <* ebwm\
         edge_based_wireframe_model.ebwm_boundary | (NOT (SIZEOF(
         QUERY ( edges <* eb.ces_edges | (NOT (
         'CONFIG_CONTROL_DESIGN.EDGE_CURVE' IN TYPEOF(edges))) ))
         = 0)) )) = 0)) )) = 0);
wr4: (SIZEOF(QUERY ( ebwm <* QUERY ( it <* SELF.items | (
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL' IN
         TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( eb <* ebwm\
         edge_based_wireframe_model.ebwm_boundary | (NOT (SIZEOF(
         QUERY ( pline_edges <* QUERY ( edges <* eb.ces_edges | (
         'CONFIG_CONTROL_DESIGN.POLYLINE' IN TYPEOF(edges\edge_curve.
         edge_geometry)) ) | (NOT (SIZEOF(pline_edges\edge_curve.
         edge_geometry\polyline.points) > 2)) )) = 0)) )) = 0)) )) =
         0);
wr5: (SIZEOF(QUERY ( ebwm <* QUERY ( it <* SELF.items | (
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL' IN
         TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( eb <* ebwm\
         edge_based_wireframe_model.ebwm_boundary | (NOT (SIZEOF(
         QUERY ( edges <* eb.ces_edges | (NOT ((
         'CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(edges.
         edge_start)) AND ('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN
         TYPEOF(edges.edge_end)))) )) = 0)) )) = 0)) )) = 0);
wr6: (SIZEOF(QUERY ( ebwm <* QUERY ( it <* SELF.items | (
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL' IN
         TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( eb <* ebwm\
         edge_based_wireframe_model.ebwm_boundary | (NOT (SIZEOF(
         QUERY ( edges <* eb.ces_edges | (NOT
         valid_wireframe_edge_curve(edges\edge_curve.edge_geometry))
         )) = 0)) )) = 0)) )) = 0);
wr7: (SIZEOF(QUERY ( ebwm <* QUERY ( it <* SELF.items | (
         'CONFIG_CONTROL_DESIGN.EDGE_BASED_WIREFRAME_MODEL' IN
         TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( eb <* ebwm\
         edge_based_wireframe_model.ebwm_boundary | (NOT (SIZEOF(
         QUERY ( edges <* eb.ces_edges | (NOT (
         valid_wireframe_vertex_point(edges.edge_start\vertex_point.
         vertex_geometry) AND valid_wireframe_vertex_point(edges.
```

```
              edge_end\vertex_point.vertex_geometry))) )) = 0)) )) = 0))
              )) = 0);
    wr8: (SIZEOF(QUERY ( mi <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (NOT
              (('CONFIG_CONTROL_DESIGN.' +
              'EDGE_BASED_WIREFRAME_SHAPE_REPRESENTATION') IN TYPEOF(mi\
              mapped_item.mapping_source.mapped_representation))) )) = 0);
    wr9: (SELF.context_of_items\geometric_representation_context.
              coordinate_space_dimension = 3);
END_ENTITY; -- edge_based_wireframe_shape_representation

ENTITY edge_curve
  SUBTYPE OF (edge, geometric_representation_item);
    edge_geometry : curve;
    same_sense    : BOOLEAN;
END_ENTITY; -- edge_curve

ENTITY edge_loop
  SUBTYPE OF (loop, path);
  DERIVE
    ne : INTEGER := SIZEOF(SELF\path.edge_list);
  WHERE
    wr1: (SELF\path.edge_list[1].edge_start :=: SELF\path.edge_list[ne].
              edge_end);
END_ENTITY; -- edge_loop

ENTITY effectivity
  SUPERTYPE OF (ONEOF (serial_numbered_effectivity,dated_effectivity,
      lot_effectivity));
    id : identifier;
END_ENTITY; -- effectivity

ENTITY elementary_surface
  SUPERTYPE OF (ONEOF (plane,cylindrical_surface,conical_surface,
      spherical_surface,toroidal_surface))
  SUBTYPE OF (surface);
    position : axis2_placement_3d;
END_ENTITY; -- elementary_surface

ENTITY ellipse
  SUBTYPE OF (conic);
    semi_axis_1 : positive_length_measure;
```

```
    semi_axis_2 : positive_length_measure;
END_ENTITY; -- ellipse

ENTITY evaluated_degenerate_pcurve
  SUBTYPE OF (degenerate_pcurve);
    equivalent_point : cartesian_point;
END_ENTITY; -- evaluated_degenerate_pcurve

ENTITY executed_action
  SUBTYPE OF (action);
END_ENTITY; -- executed_action

ENTITY face
  SUPERTYPE OF (ONEOF (face_surface,oriented_face))
  SUBTYPE OF (topological_representation_item);
    bounds : SET [1:?] OF face_bound;
  WHERE
    wr1: (NOT mixed_loop_type_set(list_to_set(list_face_loops(SELF))));
    wr2: (SIZEOF(QUERY ( temp <* bounds | (
             'CONFIG_CONTROL_DESIGN.FACE_OUTER_BOUND' IN TYPEOF(temp)) ))
             <= 1);
END_ENTITY; -- face

ENTITY face_bound
  SUBTYPE OF (topological_representation_item);
    bound       : loop;
    orientation : BOOLEAN;
END_ENTITY; -- face_bound

ENTITY face_outer_bound
  SUBTYPE OF (face_bound);
END_ENTITY; -- face_outer_bound

ENTITY face_surface
  SUBTYPE OF (face, geometric_representation_item);
    face_geometry : surface;
    same_sense    : BOOLEAN;
END_ENTITY; -- face_surface

ENTITY faceted_brep
  SUBTYPE OF (manifold_solid_brep);
END_ENTITY; -- faceted_brep
```

```
ENTITY faceted_brep_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1: (SIZEOF(QUERY ( it <* items | (NOT (SIZEOF([
             'CONFIG_CONTROL_DESIGN.FACETED_BREP',
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
             'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
             1)) )) = 0);
    wr2: (SIZEOF(QUERY ( it <* items | (SIZEOF([
             'CONFIG_CONTROL_DESIGN.FACETED_BREP',
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) )) >
              0);
    wr3: (SIZEOF(QUERY ( fbrep <* QUERY ( it <* items | (
             'CONFIG_CONTROL_DESIGN.FACETED_BREP' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( csh <* msb_shells(fbrep) | (NOT (SIZEOF(
             QUERY ( fcs <* csh\connected_face_set.cfs_faces | (NOT ((
             'CONFIG_CONTROL_DESIGN.FACE_SURFACE' IN TYPEOF(fcs)) AND (
             'CONFIG_CONTROL_DESIGN.PLANE' IN TYPEOF(fcs\face_surface.
             face_geometry)) AND ('CONFIG_CONTROL_DESIGN.CARTESIAN_POINT'
             IN TYPEOF(fcs\face_surface.face_geometry\elementary_surface.
             position.location)))) )) = 0)) )) = 0)) )) = 0);
    wr4: (SIZEOF(QUERY ( fbrep <* QUERY ( it <* items | (
             'CONFIG_CONTROL_DESIGN.FACETED_BREP' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( csh <* msb_shells(fbrep) | (NOT (SIZEOF(
             QUERY ( fcs <* csh\connected_face_set.cfs_faces | (NOT (
             SIZEOF(QUERY ( bnds <* fcs.bounds | (
             'CONFIG_CONTROL_DESIGN.FACE_OUTER_BOUND' IN TYPEOF(bnds)) ))
             = 1)) )) = 0)) )) = 0)) )) = 0);
    wr5: (SIZEOF(QUERY ( msb <* QUERY ( it <* items | (
             'CONFIG_CONTROL_DESIGN.MANIFOLD_SOLID_BREP' IN TYPEOF(it)) )
              | ('CONFIG_CONTROL_DESIGN.ORIENTED_CLOSED_SHELL' IN TYPEOF(
             msb\manifold_solid_brep.outer)) )) = 0);
    wr6: (SIZEOF(QUERY ( brv <* QUERY ( it <* items | (
             'CONFIG_CONTROL_DESIGN.BREP_WITH_VOIDS' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( csh <* brv\brep_with_voids.voids | csh\
             oriented_closed_shell.orientation )) = 0)) )) = 0);
    wr7: (SIZEOF(QUERY ( mi <* QUERY ( it <* items | (
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (NOT
             ('CONFIG_CONTROL_DESIGN.FACETED_BREP_SHAPE_REPRESENTATION'
             IN TYPEOF(mi\mapped_item.mapping_source.
             mapped_representation))) )) = 0);
END_ENTITY; -- faceted_brep_shape_representation
```

```
ENTITY founded_item;
END_ENTITY; -- founded_item

ENTITY functionally_defined_transformation;
    name        : label;
    description : text;
END_ENTITY; -- functionally_defined_transformation

ENTITY geometric_curve_set
  SUBTYPE OF (geometric_set);
  WHERE
    wr1: (SIZEOF(QUERY ( temp <* SELF\geometric_set.elements | (
             'CONFIG_CONTROL_DESIGN.SURFACE' IN TYPEOF(temp)) )) = 0);
END_ENTITY; -- geometric_curve_set

ENTITY geometric_representation_context
  SUBTYPE OF (representation_context);
    coordinate_space_dimension : dimension_count;
END_ENTITY; -- geometric_representation_context

ENTITY geometric_representation_item
  SUPERTYPE OF (ONEOF (point,direction,vector,placement,
      cartesian_transformation_operator,curve,surface,edge_curve,
      face_surface,poly_loop,vertex_point,solid_model,
      shell_based_surface_model,shell_based_wireframe_model,
      edge_based_wireframe_model,geometric_set))
  SUBTYPE OF (representation_item);
  DERIVE
    dim : dimension_count := dimension_of(SELF);
  WHERE
    wr1: (SIZEOF(QUERY ( using_rep <* using_representations(SELF) | (
             NOT (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_REPRESENTATION_CONTEXT' IN
             TYPEOF(using_rep.context_of_items))) )) = 0);
END_ENTITY; -- geometric_representation_item

ENTITY geometric_set
  SUPERTYPE OF (geometric_curve_set)
  SUBTYPE OF (geometric_representation_item);
    elements : SET [1:?] OF geometric_set_select;
END_ENTITY; -- geometric_set
```

```
ENTITY geometrically_bounded_surface_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1: (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF([
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET',
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
             'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
             1)) )) = 0);
    wr2: (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF([
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET',
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) )) >
              0);
    wr3: (SIZEOF(QUERY ( mi <* QUERY ( it <* SELF.items | (
             'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (NOT
             ((('CONFIG_CONTROL_DESIGN.' +
             'GEOMETRICALLY_BOUNDED_SURFACE_SHAPE_REPRESENTATION') IN
             TYPEOF(mi\mapped_item.mapping_source.mapped_representation))
             AND (SIZEOF(QUERY ( mr_it <* mi\mapped_item.mapping_source.
             mapped_representation.items | (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET' IN TYPEOF(mr_it)) )) >
              0))) )) = 0);
    wr4: (SIZEOF(QUERY ( gs <* QUERY ( it <* SELF.items | (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( pnt <* QUERY ( gsel <* gs\geometric_set.
             elements | ('CONFIG_CONTROL_DESIGN.POINT' IN TYPEOF(gsel)) )
              | (NOT gbsf_check_point(pnt)) )) = 0)) )) = 0);
    wr5: (SIZEOF(QUERY ( gs <* QUERY ( it <* SELF.items | (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( cv <* QUERY ( gsel <* gs\geometric_set.
             elements | ('CONFIG_CONTROL_DESIGN.CURVE' IN TYPEOF(gsel)) )
              | (NOT gbsf_check_curve(cv)) )) = 0)) )) = 0);
    wr6: (SIZEOF(QUERY ( gs <* QUERY ( it <* SELF.items | (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET' IN TYPEOF(it)) ) | (
             NOT (SIZEOF(QUERY ( sf <* QUERY ( gsel <* gs\geometric_set.
             elements | ('CONFIG_CONTROL_DESIGN.SURFACE' IN
             TYPEOF(gsel)) ) | (NOT gbsf_check_surface(sf)) ))
             = 0)) )) = 0);
    wr7: (SIZEOF(QUERY ( gs <* QUERY ( it <* SELF.items | (
             'CONFIG_CONTROL_DESIGN.GEOMETRIC_SET' IN TYPEOF(it)) ) | (
             SIZEOF(QUERY ( gsel <* gs\geometric_set.elements | (
```

```
              'CONFIG_CONTROL_DESIGN.SURFACE' IN TYPEOF(gsel)) )) > 0) ))
              > 0);
END_ENTITY; -- geometrically_bounded_surface_shape_representation

ENTITY geometrically_bounded_wireframe_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1: (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF(TYPEOF(it) * [
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET',
              'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D',
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM']) = 1)) )) = 0);
    wr2: (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF(TYPEOF(it) * [
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET',
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM']) = 1) )) >= 1);
    wr3: (SIZEOF(QUERY ( gcs <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET' IN TYPEOF(it)) )
               | (NOT (SIZEOF(QUERY ( crv <* QUERY ( elem <* gcs\
              geometric_set.elements | ('CONFIG_CONTROL_DESIGN.CURVE' IN
              TYPEOF(elem)) ) | (NOT valid_geometrically_bounded_wf_curve(
              crv)) )) = 0)) )) = 0);
    wr4: (SIZEOF(QUERY ( gcs <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET' IN TYPEOF(it)) )
               | (NOT (SIZEOF(QUERY ( pnts <* QUERY ( elem <* gcs\
              geometric_set.elements | ('CONFIG_CONTROL_DESIGN.POINT' IN
              TYPEOF(elem)) ) | (NOT valid_geometrically_bounded_wf_point(
              pnts)) )) = 0)) )) = 0);
    wr5: (SIZEOF(QUERY ( gcs <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET' IN TYPEOF(it)) )
               | (NOT (SIZEOF(QUERY ( cnc <* QUERY ( elem <* gcs\
              geometric_set.elements | ('CONFIG_CONTROL_DESIGN.CONIC' IN
              TYPEOF(elem)) ) | (NOT (
              'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D' IN TYPEOF(cnc\
              conic.position))) )) = 0)) )) = 0);
    wr6: (SIZEOF(QUERY ( gcs <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.GEOMETRIC_CURVE_SET' IN TYPEOF(it)) )
               | (NOT (SIZEOF(QUERY ( pline <* QUERY ( elem <* gcs\
              geometric_set.elements | ('CONFIG_CONTROL_DESIGN.POLYLINE'
              IN TYPEOF(elem)) ) | (NOT (SIZEOF(pline\polyline.points)
              > 2)) )) = 0)) )) = 0);
    wr7: (SIZEOF(QUERY ( mi <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (NOT
              (('CONFIG_CONTROL_DESIGN.' +
```

```
              'GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION') IN
              TYPEOF(mi\mapped_item.mapping_source.mapped_representation)
              )) )) = 0);
END_ENTITY; -- geometrically_bounded_wireframe_shape_representation

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

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

ENTITY hyperbola
  SUBTYPE OF (conic);
    semi_axis      : positive_length_measure;
    semi_imag_axis : positive_length_measure;
END_ENTITY; -- hyperbola

ENTITY intersection_curve
  SUBTYPE OF (surface_curve);
  WHERE
    wr1: (SIZEOF(SELF\surface_curve.associated_geometry) = 2);
    wr2: (associated_surface(SELF\surface_curve.associated_geometry[1])
              <> associated_surface(SELF\surface_curve.associated_geometry
              [2]));
END_ENTITY; -- intersection_curve

ENTITY item_defined_transformation;
    name             : label;
    description      : text;
    transform_item_1 : representation_item;
    transform_item_2 : representation_item;
END_ENTITY; -- item_defined_transformation

ENTITY length_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.LENGTH_UNIT' IN TYPEOF(SELF\
```

```
              measure_with_unit.unit_component));
END_ENTITY; -- length_measure_with_unit

ENTITY length_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 1) AND (SELF\
              named_unit.dimensions.mass_exponent = 0) AND (SELF\
              named_unit.dimensions.time_exponent = 0) AND (SELF\
              named_unit.dimensions.electric_current_exponent = 0) AND (
              SELF\named_unit.dimensions.
              thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
              .dimensions.amount_of_substance_exponent = 0) AND (SELF\
              named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- length_unit

ENTITY line
  SUBTYPE OF (curve);
    pnt : cartesian_point;
    dir : vector;
  WHERE
    wr1: (dir.dim = pnt.dim);
END_ENTITY; -- line

ENTITY local_time;
    hour_component   : hour_in_day;
    minute_component : OPTIONAL minute_in_hour;
    second_component : OPTIONAL second_in_minute;
    zone             : coordinated_universal_time_offset;
  WHERE
    wr1: valid_time(SELF);
END_ENTITY; -- local_time

ENTITY loop
  SUPERTYPE OF (ONEOF (vertex_loop,edge_loop,poly_loop))
  SUBTYPE OF (topological_representation_item);
END_ENTITY; -- loop

ENTITY lot_effectivity
  SUBTYPE OF (effectivity);
    effectivity_lot_id   : identifier;
```

```
    effectivity_lot_size : measure_with_unit;
END_ENTITY; -- lot_effectivity

ENTITY manifold_solid_brep
  SUBTYPE OF (solid_model);
    outer : closed_shell;
END_ENTITY; -- manifold_solid_brep

ENTITY manifold_surface_shape_representation
  SUBTYPE OF (shape_representation);
  WHERE
    wr1 : (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF([
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL',
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
              'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
               1)) )) = 0);
    wr2 : (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF([
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL',
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) ))
              > 0);
    wr3 : (SIZEOF(QUERY ( mi <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (
              NOT (('CONFIG_CONTROL_DESIGN.'+
             'MANIFOLD_SURFACE_SHAPE_REPRESENTATION'
              IN TYPEOF(mi\mapped_item.mapping_source.
              mapped_representation)) AND (SIZEOF(QUERY ( mr_it <* mi\
              mapped_item.mapping_source.mapped_representation.items | (
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
              TYPEOF(mr_it)) )) > 0))) )) = 0);
    wr4 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
              TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( sh <* sbsm\
              shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF([
              'CONFIG_CONTROL_DESIGN.OPEN_SHELL',
              'CONFIG_CONTROL_DESIGN.ORIENTED_CLOSED_SHELL',
              'CONFIG_CONTROL_DESIGN.CLOSED_SHELL'] * TYPEOF(sh)) = 1))
              )) = 0)) )) = 0);
    wr5 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
              TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
              shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
              QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT (
```

```
          SIZEOF(['CONFIG_CONTROL_DESIGN.FACE_SURFACE',
          'CONFIG_CONTROL_DESIGN.ORIENTED_FACE'] * TYPEOF(fa)) = 1))
          )) = 0)) )) = 0)) )) = 0);
wr6 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
          shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
          QUERY ( f_sf <* QUERY ( fa <* cfs\connected_face_set.
          cfs_faces | ('CONFIG_CONTROL_DESIGN.FACE_SURFACE' IN
          TYPEOF(fa)) ) | (NOT ((
          'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(f_sf)) OR (
          SIZEOF(['CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE',
          'CONFIG_CONTROL_DESIGN.ELEMENTARY_SURFACE',
          'CONFIG_CONTROL_DESIGN.OFFSET_SURFACE',
          'CONFIG_CONTROL_DESIGN.SURFACE_REPLICA',
          'CONFIG_CONTROL_DESIGN.SWEPT_SURFACE'] * TYPEOF(f_sf\
          face_surface.face_geometry)) = 1))) )) = 0)) )) = 0)) )) =
          0);
wr7 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
          shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
          QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
          'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR
          msf_surface_check(fa\face_surface.face_geometry))) ))
          = 0)) )) = 0)) )) = 0);
wr8 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
          shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
          QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
          'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
          SIZEOF(QUERY ( bnds <* fa.bounds | (NOT (SIZEOF([
          'CONFIG_CONTROL_DESIGN.EDGE_LOOP',
          'CONFIG_CONTROL_DESIGN.VERTEX_LOOP'] * TYPEOF(bnds.bound))
          = 1)) )) = 0))) )) = 0)) )) = 0)) )) = 0);
wr9 : (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
          shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
          QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
          'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
```

```
            SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
             | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds\path.edge_list | (
            NOT ('CONFIG_CONTROL_DESIGN.EDGE_CURVE' IN TYPEOF(oe.
            edge_element))))) = 0)))) = 0))))) = 0)) )) = 0)) )) = 0);
wr10: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
            shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
            QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
            'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
            SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
             | (NOT (SIZEOF(QUERY ( oe_cv <* QUERY ( oe <* elp_fbnds\
            path.edge_list | ('CONFIG_CONTROL_DESIGN.EDGE_CURVE' IN
            TYPEOF(oe.edge_element)) ) | (NOT (SIZEOF([
            'CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE',
            'CONFIG_CONTROL_DESIGN.CONIC',
            'CONFIG_CONTROL_DESIGN.CURVE_REPLICA',
            'CONFIG_CONTROL_DESIGN.LINE',
            'CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D',
            'CONFIG_CONTROL_DESIGN.PCURVE',
            'CONFIG_CONTROL_DESIGN.POLYLINE',
            'CONFIG_CONTROL_DESIGN.SURFACE_CURVE'] * TYPEOF(oe_cv.
            edge_element\edge_curve.edge_geometry)) = 1)) )) = 0)) )) =
             0))) )) = 0)) )) = 0)) )) = 0);
wr11: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
            shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
            QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
            'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
            SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
             | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds\path.edge_list | (
            NOT msf_curve_check(oe.edge_element\edge_curve.
            edge_geometry)) )) = 0)) )) = 0))) ))
            = 0)) )) = 0)) )) = 0);
wr12: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
            shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
```

```
            QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
            'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
            SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
             | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds\path.edge_list | (
            NOT (('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(oe.
            edge_element.edge_start)) AND (
            'CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(oe.
            edge_element.edge_end)))) )) = 0)) )) = 0))) )) = 0)) )) =
            0)) )) = 0);
wr13: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
            shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
            QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
            'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
            SIZEOF(QUERY ( elp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN TYPEOF(bnds.bound)) )
             | (NOT (SIZEOF(QUERY ( oe <* elp_fbnds\path.edge_list | (
            NOT ((SIZEOF(['CONFIG_CONTROL_DESIGN.CARTESIAN_POINT',
            'CONFIG_CONTROL_DESIGN.DEGENERATE_PCURVE',
            'CONFIG_CONTROL_DESIGN.POINT_ON_CURVE',
            'CONFIG_CONTROL_DESIGN.POINT_ON_SURFACE'] * TYPEOF(oe.
            edge_element.edge_start\vertex_point.vertex_geometry)) = 1)
            AND (SIZEOF(['CONFIG_CONTROL_DESIGN.CARTESIAN_POINT',
            'CONFIG_CONTROL_DESIGN.DEGENERATE_PCURVE',
            'CONFIG_CONTROL_DESIGN.POINT_ON_CURVE',
            'CONFIG_CONTROL_DESIGN.POINT_ON_SURFACE'] * TYPEOF(oe.
            edge_element.edge_end\vertex_point.vertex_geometry))
            = 1))) )) = 0)) )) = 0))) )) = 0)) )) = 0)) )) = 0);
wr14: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
            shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
            QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
            'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
            SIZEOF(QUERY ( vlp_fbnds <* QUERY ( bnds <* fa.bounds | (
            'CONFIG_CONTROL_DESIGN.VERTEX_LOOP'
            IN TYPEOF(bnds.bound)) )
             | (NOT ('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(
            vlp_fbnds\vertex_loop.loop_vertex))) )) = 0))) )) = 0)) ))
            = 0)) )) = 0);
```

```
    wr15: (SIZEOF(QUERY ( sbsm <* QUERY ( it <* SELF.items | (
              'CONFIG_CONTROL_DESIGN.SHELL_BASED_SURFACE_MODEL' IN
              TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( cfs <* sbsm\
              shell_based_surface_model.sbsm_boundary | (NOT (SIZEOF(
              QUERY ( fa <* cfs\connected_face_set.cfs_faces | (NOT ((
              'CONFIG_CONTROL_DESIGN.ADVANCED_FACE' IN TYPEOF(fa)) OR (
              SIZEOF(QUERY ( vlp_fbnds <* QUERY ( bnds <* fa.bounds | (
              'CONFIG_CONTROL_DESIGN.VERTEX_LOOP'
              IN TYPEOF(bnds.bound)) )
               | (NOT (SIZEOF(['CONFIG_CONTROL_DESIGN.CARTESIAN_POINT',
              'CONFIG_CONTROL_DESIGN.DEGENERATE_PCURVE',
              'CONFIG_CONTROL_DESIGN.POINT_ON_CURVE',
              'CONFIG_CONTROL_DESIGN.POINT_ON_SURFACE'] * TYPEOF(
              vlp_fbnds\vertex_loop.loop_vertex\vertex_point.
              vertex_geometry)) = 1)) )) = 0))) ))
              = 0)) )) = 0)) )) = 0);
END_ENTITY; -- manifold_surface_shape_representation

ENTITY mapped_item
  SUBTYPE OF (representation_item);
    mapping_source : representation_map;
    mapping_target : representation_item;
  WHERE
    wr1: acyclic_mapped_representation(using_representations(SELF),
         [SELF]);
END_ENTITY; -- mapped_item

ENTITY mass_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.MASS_UNIT' IN TYPEOF(SELF\
             measure_with_unit.unit_component));
END_ENTITY; -- mass_measure_with_unit

ENTITY mass_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 0) AND (SELF\
             named_unit.dimensions.mass_exponent = 1) AND (SELF\
             named_unit.dimensions.time_exponent = 0) AND (SELF\
             named_unit.dimensions.electric_current_exponent = 0) AND (
             SELF\named_unit.dimensions.
```

```
              thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
              .dimensions.amount_of_substance_exponent = 0) AND (SELF\
              named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- mass_unit

ENTITY measure_with_unit
  SUPERTYPE OF (ONEOF (length_measure_with_unit,mass_measure_with_unit,
      plane_angle_measure_with_unit,solid_angle_measure_with_unit,
      area_measure_with_unit,volume_measure_with_unit));
    value_component : measure_value;
    unit_component  : unit;
  WHERE
    wr1: valid_units(SELF);
END_ENTITY; -- measure_with_unit

ENTITY mechanical_context
  SUBTYPE OF (product_context);
  WHERE
    wr1: (SELF.discipline_type = 'mechanical');
END_ENTITY; -- mechanical_context

ENTITY named_unit
  SUPERTYPE OF (ONEOF (si_unit,conversion_based_unit,
      context_dependent_unit) ANDOR ONEOF (length_unit,mass_unit,
      plane_angle_unit,solid_angle_unit,area_unit,volume_unit));
    dimensions : dimensional_exponents;
END_ENTITY; -- named_unit

ENTITY next_assembly_usage_occurrence
  SUBTYPE OF (assembly_component_usage);
END_ENTITY; -- next_assembly_usage_occurrence

ENTITY offset_curve_3d
  SUBTYPE OF (curve);
    basis_curve    : curve;
    distance       : length_measure;
    self_intersect : LOGICAL;
    ref_direction  : direction;
  WHERE
    wr1: ((basis_curve.dim = 3) AND (ref_direction.dim = 3));
END_ENTITY; -- offset_curve_3d
```

```
ENTITY offset_surface
  SUBTYPE OF (surface);
    basis_surface  : surface;
    distance       : length_measure;
    self_intersect : LOGICAL;
END_ENTITY; -- offset_surface

ENTITY open_shell
  SUBTYPE OF (connected_face_set);
END_ENTITY; -- open_shell

ENTITY ordinal_date
  SUBTYPE OF (date);
    day_component : day_in_year_number;
  WHERE
    wr1: (((NOT leap_year(SELF.year_component)) AND (1 <= day_component)
             AND (day_component <= 365)) OR (leap_year(SELF.
             year_component) AND (1 <= day_component) AND (day_component
             <= 366)));
END_ENTITY; -- ordinal_date

ENTITY organization;
    id          : OPTIONAL identifier;
    name        : label;
    description : text;
END_ENTITY; -- organization

ENTITY organization_relationship;
    name                  : label;
    description           : text;
    relating_organization : organization;
    related_organization  : organization;
END_ENTITY; -- organization_relationship

ENTITY organizational_address
  SUBTYPE OF (address);
    organizations : SET [1:?] OF organization;
    description   : text;
END_ENTITY; -- organizational_address

ENTITY organizational_project;
    name                      : label;
```

```
    description               : text;
    responsible_organizations : SET [1:?] OF organization;
END_ENTITY; -- organizational_project

ENTITY oriented_closed_shell
  SUBTYPE OF (closed_shell);
    closed_shell_element : closed_shell;
    orientation          : BOOLEAN;
  DERIVE
    SELF\connected_face_set.cfs_faces : SET [1:?] OF face :=
        conditional_reverse(SELF.orientation,SELF.
        closed_shell_element.cfs_faces);
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.ORIENTED_CLOSED_SHELL' IN TYPEOF(
             SELF.closed_shell_element)));
END_ENTITY; -- oriented_closed_shell

ENTITY oriented_edge
  SUBTYPE OF (edge);
    edge_element : edge;
    orientation  : BOOLEAN;
  DERIVE
    SELF\edge.edge_start : vertex := boolean_choose(SELF.orientation,
                               SELF.edge_element.edge_start,SELF.
                               edge_element.edge_end);
    SELF\edge.edge_end   : vertex := boolean_choose(SELF.orientation,
                               SELF.edge_element.edge_end,SELF.
                               edge_element.edge_start);
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.ORIENTED_EDGE' IN TYPEOF(SELF.
             edge_element)));
END_ENTITY; -- oriented_edge

ENTITY oriented_face
  SUBTYPE OF (face);
    face_element : face;
    orientation  : BOOLEAN;
  DERIVE
    SELF\face.bounds : SET [1:?] OF face_bound := conditional_reverse(
                           SELF.orientation,SELF.face_element.bounds);
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.ORIENTED_FACE' IN TYPEOF(SELF.
```

```
              face_element)));
END_ENTITY; -- oriented_face

ENTITY oriented_open_shell
  SUBTYPE OF (open_shell);
    open_shell_element : open_shell;
    orientation        : BOOLEAN;
  DERIVE
    SELF\connected_face_set.cfs_faces : SET [1:?] OF face :=
                                          conditional_reverse(SELF.
                                          orientation,SELF.
                                          open_shell_element.cfs_faces);
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.ORIENTED_OPEN_SHELL' IN TYPEOF(
              SELF.open_shell_element)));
END_ENTITY; -- oriented_open_shell

ENTITY oriented_path
  SUBTYPE OF (path);
    path_element : path;
    orientation  : BOOLEAN;
  DERIVE
    SELF\path.edge_list : LIST [1:?] OF UNIQUE oriented_edge :=
                              conditional_reverse(SELF.orientation,SELF.
                              path_element.edge_list);
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.ORIENTED_PATH' IN TYPEOF(SELF.
              path_element)));
END_ENTITY; -- oriented_path

ENTITY outer_boundary_curve
  SUBTYPE OF (boundary_curve);
END_ENTITY; -- outer_boundary_curve

ENTITY parabola
  SUBTYPE OF (conic);
    focal_dist : length_measure;
  WHERE
    wr1: (focal_dist <> 0);
END_ENTITY; -- parabola
```

```
ENTITY parametric_representation_context
  SUBTYPE OF (representation_context);
END_ENTITY; -- parametric_representation_context

ENTITY path
  SUPERTYPE OF (ONEOF (edge_loop,oriented_path))
  SUBTYPE OF (topological_representation_item);
    edge_list : LIST [1:?] OF UNIQUE oriented_edge;
  WHERE
    wr1: path_head_to_tail(SELF);
END_ENTITY; -- path

ENTITY pcurve
  SUBTYPE OF (curve);
    basis_surface      : surface;
    reference_to_curve : definitional_representation;
  WHERE
    wr1: (SIZEOF(reference_to_curve\representation.items) = 1);
    wr2: ('CONFIG_CONTROL_DESIGN.CURVE' IN TYPEOF(reference_to_curve\
             representation.items[1]));
    wr3: (reference_to_curve\representation.items[1]\
             geometric_representation_item.dim = 2);
END_ENTITY; -- pcurve

ENTITY person;
    id            : identifier;
    last_name     : OPTIONAL label;
    first_name    : OPTIONAL label;
    middle_names  : OPTIONAL LIST [1:?] OF label;
    prefix_titles : OPTIONAL LIST [1:?] OF label;
    suffix_titles : OPTIONAL LIST [1:?] OF label;
  UNIQUE
    ur1 : id;
  WHERE
    wr1: (EXISTS(last_name) OR EXISTS(first_name));
END_ENTITY; -- person

ENTITY person_and_organization;
    the_person       : person;
    the_organization : organization;
END_ENTITY; -- person_and_organization
```

```
ENTITY person_and_organization_assignment
  ABSTRACT SUPERTYPE;
    assigned_person_and_organization : person_and_organization;
    role                             : person_and_organization_role;
END_ENTITY; -- person_and_organization_assignment

ENTITY person_and_organization_role;
    name : label;
END_ENTITY; -- person_and_organization_role

ENTITY personal_address
  SUBTYPE OF (address);
    people      : SET [1:?] OF person;
    description : text;
END_ENTITY; -- personal_address

ENTITY placement
  SUPERTYPE OF (ONEOF (axis1_placement,axis2_placement_2d,
      axis2_placement_3d))
  SUBTYPE OF (geometric_representation_item);
    location : cartesian_point;
END_ENTITY; -- placement

ENTITY plane
  SUBTYPE OF (elementary_surface);
END_ENTITY; -- plane

ENTITY plane_angle_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.PLANE_ANGLE_UNIT' IN TYPEOF(SELF\
             measure_with_unit.unit_component));
END_ENTITY; -- plane_angle_measure_with_unit

ENTITY plane_angle_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 0) AND (SELF\
             named_unit.dimensions.mass_exponent = 0) AND (SELF\
             named_unit.dimensions.time_exponent = 0) AND (SELF\
             named_unit.dimensions.electric_current_exponent = 0) AND (
             SELF\named_unit.dimensions.
```

```
              thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
              .dimensions.amount_of_substance_exponent = 0) AND (SELF\
              named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- plane_angle_unit

ENTITY point
  SUPERTYPE OF (ONEOF (cartesian_point,point_on_curve,point_on_surface,
      point_replica,degenerate_pcurve))
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- point

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

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

ENTITY point_replica
  SUBTYPE OF (point);
    parent_pt      : point;
    transformation : cartesian_transformation_operator;
  WHERE
    wr1: (transformation.dim = parent_pt.dim);
    wr2: acyclic_point_replica(SELF,parent_pt);
END_ENTITY; -- point_replica

ENTITY poly_loop
  SUBTYPE OF (loop, geometric_representation_item);
    polygon : LIST [3:?] OF UNIQUE cartesian_point;
END_ENTITY; -- poly_loop

ENTITY polyline
  SUBTYPE OF (bounded_curve);
    points : LIST [2:?] OF cartesian_point;
END_ENTITY; -- polyline
```

```
ENTITY product;
    id                 : identifier;
    name               : label;
    description        : text;
    frame_of_reference : SET [1:?] OF product_context;
  UNIQUE
    ur1 : id;
END_ENTITY; -- product

ENTITY product_category;
    name        : label;
    description : OPTIONAL text;
END_ENTITY; -- product_category

ENTITY product_category_relationship;
    name         : label;
    description  : text;
    category     : product_category;
    sub_category : product_category;
  WHERE
    wr1: acyclic_product_category_relationship(SELF,[SELF.sub_category]);
END_ENTITY; -- product_category_relationship

ENTITY product_concept;
    id             : identifier;
    name           : label;
    description    : text;
    market_context : product_concept_context;
  UNIQUE
    ur1 : id;
END_ENTITY; -- product_concept

ENTITY product_concept_context
  SUBTYPE OF (application_context_element);
    market_segment_type : label;
END_ENTITY; -- product_concept_context

ENTITY product_context
  SUBTYPE OF (application_context_element);
    discipline_type : label;
END_ENTITY; -- product_context
```

```
ENTITY product_definition;
    id                 : identifier;
    description        : text;
    formation          : product_definition_formation;
    frame_of_reference : product_definition_context;
END_ENTITY; -- product_definition

ENTITY product_definition_context
  SUBTYPE OF (application_context_element);
    life_cycle_stage : label;
END_ENTITY; -- product_definition_context

ENTITY product_definition_effectivity
  SUBTYPE OF (effectivity);
    usage : product_definition_relationship;
  UNIQUE
    ur1 : usage, id;
END_ENTITY; -- product_definition_effectivity

ENTITY product_definition_formation;
    id          : identifier;
    description : text;
    of_product  : product;
  UNIQUE
    ur1 : id, of_product;
END_ENTITY; -- product_definition_formation

ENTITY product_definition_formation_with_specified_source
  SUBTYPE OF (product_definition_formation);
    make_or_buy : source;
END_ENTITY; -- product_definition_formation_with_specified_source

ENTITY product_definition_relationship;
    id                          : identifier;
    name                        : label;
    description                 : text;
    relating_product_definition : product_definition;
    related_product_definition  : product_definition;
END_ENTITY; -- product_definition_relationship

ENTITY product_definition_shape
  SUBTYPE OF (property_definition);
```

```
    UNIQUE
      ur1 : definition;
    WHERE
      wr1: (NOT ('CONFIG_CONTROL_DESIGN.SHAPE_DEFINITION' IN TYPEOF(SELF\
               property_definition.definition)));
  END_ENTITY; -- product_definition_shape

  ENTITY product_definition_usage
    SUPERTYPE OF (assembly_component_usage)
    SUBTYPE OF (product_definition_relationship);
    UNIQUE
      ur1 : id, relating_product_definition, related_product_definition;
    WHERE
      wr1: acyclic_product_definition_relationship(SELF,[SELF\
               product_definition_relationship.related_product_definition],
               'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_USAGE');
  END_ENTITY; -- product_definition_usage

  ENTITY product_definition_with_associated_documents
    SUBTYPE OF (product_definition);
      documentation_ids : SET [1:?] OF document;
  END_ENTITY; -- product_definition_with_associated_documents

  ENTITY product_related_product_category
    SUBTYPE OF (product_category);
      products : SET [1:?] OF product;
  END_ENTITY; -- product_related_product_category

  ENTITY promissory_usage_occurrence
    SUBTYPE OF (assembly_component_usage);
  END_ENTITY; -- promissory_usage_occurrence

  ENTITY property_definition;
      name        : label;
      description : text;
      definition  : characterized_definition;
  END_ENTITY; -- property_definition

  ENTITY property_definition_representation;
      definition          : property_definition;
      used_representation : representation;
  END_ENTITY; -- property_definition_representation
```

```
ENTITY quantified_assembly_component_usage
  SUBTYPE OF (assembly_component_usage);
    quantity : measure_with_unit;
END_ENTITY; -- quantified_assembly_component_usage

ENTITY quasi_uniform_curve
  SUBTYPE OF (b_spline_curve);
END_ENTITY; -- quasi_uniform_curve

ENTITY quasi_uniform_surface
  SUBTYPE OF (b_spline_surface);
END_ENTITY; -- quasi_uniform_surface

ENTITY rational_b_spline_curve
  SUBTYPE OF (b_spline_curve);
    weights_data : LIST [2:?] OF REAL;
  DERIVE
    weights : ARRAY [0:upper_index_on_control_points] OF REAL :=
                 list_to_array(weights_data,0,
                 upper_index_on_control_points);
  WHERE
    wr1: (SIZEOF(weights_data) = SIZEOF(SELF\b_spline_curve.
             control_points_list));
    wr2: curve_weights_positive(SELF);
END_ENTITY; -- rational_b_spline_curve

ENTITY rational_b_spline_surface
  SUBTYPE OF (b_spline_surface);
    weights_data : LIST [2:?] OF LIST [2:?] OF REAL;
  DERIVE
    weights : ARRAY [0:u_upper] OF ARRAY [0:v_upper] OF REAL :=
                 make_array_of_array(weights_data,0,u_upper,0,v_upper);
  WHERE
    wr1: ((SIZEOF(weights_data) = SIZEOF(SELF\b_spline_surface.
             control_points_list)) AND (SIZEOF(weights_data[1]) = SIZEOF(
             SELF\b_spline_surface.control_points_list[1])));
    wr2: surface_weights_positive(SELF);
END_ENTITY; -- rational_b_spline_surface

ENTITY rectangular_composite_surface
  SUBTYPE OF (bounded_surface);
    segments : LIST [1:?] OF LIST [1:?] OF surface_patch;
```

```
  DERIVE
    n_u : INTEGER := SIZEOF(segments);
    n_v : INTEGER := SIZEOF(segments[1]);
  WHERE
    wr1: ([] = QUERY ( s <* segments | (n_v <> SIZEOF(s)) ));
    wr2: constraints_rectangular_composite_surface(SELF);
END_ENTITY; -- rectangular_composite_surface

ENTITY rectangular_trimmed_surface
  SUBTYPE OF (bounded_surface);
    basis_surface : surface;
    u1            : parameter_value;
    u2            : parameter_value;
    v1            : parameter_value;
    v2            : parameter_value;
    usense        : BOOLEAN;
    vsense        : BOOLEAN;
  WHERE
    wr1: (u1 <> u2);
    wr2: (v1 <> v2);
    wr3: ((('CONFIG_CONTROL_DESIGN.ELEMENTARY_SURFACE' IN TYPEOF(
             basis_surface)) AND (NOT ('CONFIG_CONTROL_DESIGN.PLANE' IN
             TYPEOF(basis_surface)))) OR (
             'CONFIG_CONTROL_DESIGN.SURFACE_OF_REVOLUTION' IN TYPEOF(
             basis_surface)) OR (usense = (u2 > u1)));
    wr4: (('CONFIG_CONTROL_DESIGN.SPHERICAL_SURFACE' IN TYPEOF(
             basis_surface)) OR ('CONFIG_CONTROL_DESIGN.TOROIDAL_SURFACE'
             IN TYPEOF(basis_surface)) OR (vsense = (v2 > v1)));
END_ENTITY; -- rectangular_trimmed_surface

ENTITY reparametrised_composite_curve_segment
  SUBTYPE OF (composite_curve_segment);
    param_length : parameter_value;
  WHERE
    wr1: (param_length > 0);
END_ENTITY; -- reparametrised_composite_curve_segment

ENTITY representation;
    name             : label;
    items            : SET [1:?] OF representation_item;
    context_of_items : representation_context;
END_ENTITY; -- representation
```

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

ENTITY representation_item;
    name : label;
  WHERE
    wr1: (SIZEOF(using_representations(SELF)) > 0);
END_ENTITY; -- representation_item

ENTITY representation_map;
    mapping_origin        : representation_item;
    mapped_representation : representation;
  INVERSE
    map_usage : SET [1:?] OF mapped_item FOR mapping_source;
  WHERE
    wr1: item_in_context(SELF.mapping_origin,SELF.mapped_representation.
             context_of_items);
END_ENTITY; -- representation_map

ENTITY representation_relationship;
    name        : label;
    description : text;
    rep_1       : representation;
    rep_2       : representation;
END_ENTITY; -- representation_relationship

ENTITY representation_relationship_with_transformation
  SUBTYPE OF (representation_relationship);
    transformation_operator : transformation;
  WHERE
    wr1: (SELF\representation_relationship.rep_1.context_of_items :<>:
             SELF\representation_relationship.rep_2.context_of_items);
END_ENTITY; -- representation_relationship_with_transformation

ENTITY seam_curve
  SUBTYPE OF (surface_curve);
  WHERE
```

```
    wr1: (SIZEOF(SELF\surface_curve.associated_geometry) = 2);
    wr2: (associated_surface(SELF\surface_curve.associated_geometry[1])
             = associated_surface(SELF\surface_curve.associated_geometry[
             2]));
    wr3: ('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(SELF\surface_curve.
             associated_geometry[1]));
    wr4: ('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(SELF\surface_curve.
             associated_geometry[2]));
END_ENTITY; -- seam_curve

ENTITY security_classification;
    name           : label;
    purpose        : text;
    security_level : security_classification_level;
END_ENTITY; -- security_classification

ENTITY security_classification_assignment
  ABSTRACT SUPERTYPE;
    assigned_security_classification : security_classification;
END_ENTITY; -- security_classification_assignment

ENTITY security_classification_level;
    name : label;
END_ENTITY; -- security_classification_level

ENTITY serial_numbered_effectivity
  SUBTYPE OF (effectivity);
    effectivity_start_id : identifier;
    effectivity_end_id   : OPTIONAL identifier;
END_ENTITY; -- serial_numbered_effectivity

ENTITY shape_aspect;
    name                 : label;
    description          : text;
    of_shape             : product_definition_shape;
    product_definitional : LOGICAL;
END_ENTITY; -- shape_aspect

ENTITY shape_aspect_relationship;
    name                  : label;
    description           : text;
    relating_shape_aspect : shape_aspect;
```

```
    related_shape_aspect  : shape_aspect;
END_ENTITY; -- shape_aspect_relationship

ENTITY shape_definition_representation
  SUBTYPE OF (property_definition_representation);
  WHERE
    wr1: (('CONFIG_CONTROL_DESIGN.SHAPE_DEFINITION' IN TYPEOF(SELF.
             definition.definition)) OR (
             'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_SHAPE' IN TYPEOF(
             SELF.definition)));
    wr2: ('CONFIG_CONTROL_DESIGN.SHAPE_REPRESENTATION' IN TYPEOF(SELF.
             used_representation));
END_ENTITY; -- shape_definition_representation

ENTITY shape_representation
  SUBTYPE OF (representation);
END_ENTITY; -- shape_representation

ENTITY shape_representation_relationship
  SUBTYPE OF (representation_relationship);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.SHAPE_REPRESENTATION' IN (TYPEOF(SELF\
             representation_relationship.rep_1) + TYPEOF(SELF\
             representation_relationship.rep_2)));
END_ENTITY; -- shape_representation_relationship

ENTITY shell_based_surface_model
  SUBTYPE OF (geometric_representation_item);
    sbsm_boundary : SET [1:?] OF shell;
  WHERE
    wr1: constraints_geometry_shell_based_surface_model(SELF);
END_ENTITY; -- shell_based_surface_model

ENTITY shell_based_wireframe_model
  SUBTYPE OF (geometric_representation_item);
    sbwm_boundary : SET [1:?] OF shell;
  WHERE
    wr1: constraints_geometry_shell_based_wireframe_model(SELF);
END_ENTITY; -- shell_based_wireframe_model

ENTITY shell_based_wireframe_shape_representation
  SUBTYPE OF (shape_representation);
```

```
WHERE
  wr1 : (SIZEOF(QUERY ( it <* SELF.items | (NOT (SIZEOF([
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL',
            'CONFIG_CONTROL_DESIGN.MAPPED_ITEM',
            'CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D'] * TYPEOF(it)) =
             1)) )) = 0);
  wr2 : (SIZEOF(QUERY ( it <* SELF.items | (SIZEOF([
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL',
            'CONFIG_CONTROL_DESIGN.MAPPED_ITEM'] * TYPEOF(it)) = 1) ))
            >= 1);
  wr3 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
            sbwm\shell_based_wireframe_model.sbwm_boundary | (
            'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
            (SIZEOF(QUERY ( eloop <* QUERY ( wsb <* ws\wire_shell.
            wire_shell_extent | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN
            TYPEOF(wsb)) ) | (NOT (SIZEOF(QUERY ( el <* eloop\path.
            edge_list | (NOT ('CONFIG_CONTROL_DESIGN.EDGE_CURVE' IN
            TYPEOF(el.edge_element))) )) = 0)) ))
            = 0)) )) = 0)) )) = 0);
  wr4 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
            sbwm\shell_based_wireframe_model.sbwm_boundary | (
            'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
            (SIZEOF(QUERY ( eloop <* QUERY ( wsb <* ws\wire_shell.
            wire_shell_extent | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN
            TYPEOF(wsb)) ) | (NOT (SIZEOF(QUERY ( pline_el <*
            QUERY ( el <* eloop\path.edge_list | (
            'CONFIG_CONTROL_DESIGN.POLYLINE' IN TYPEOF(el.edge_element\
            edge_curve.edge_geometry)) ) | (NOT (SIZEOF(pline_el.
            edge_element\edge_curve.edge_geometry\polyline.points)
            > 2)) )) = 0)) )) = 0)) )) = 0)) )) = 0);
  wr5 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
            'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
            TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
            sbwm\shell_based_wireframe_model.sbwm_boundary | (
            'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
            (SIZEOF(QUERY ( eloop <* QUERY ( wsb <* ws\wire_shell.
            wire_shell_extent | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN
            TYPEOF(wsb)) ) | (NOT (SIZEOF(QUERY ( el <* eloop\path.
```

```
          edge_list | (NOT valid_wireframe_edge_curve(el.edge_element
          \edge_curve.edge_geometry)) )) = 0)) )) = 0)) )) = 0)) )) =
           0);
wr6 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
          sbwm\shell_based_wireframe_model.sbwm_boundary | (
          'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
          (SIZEOF(QUERY ( eloop <* QUERY ( wsb <* ws\wire_shell.
          wire_shell_extent | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN
          TYPEOF(wsb)) ) | (NOT (SIZEOF(QUERY ( el <* eloop\path.
          edge_list | (NOT (('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN
          TYPEOF(el.edge_element.edge_start)) AND (
          'CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(el.
          edge_element.edge_end)))) )) = 0)) ))
          = 0)) )) = 0)) )) = 0);
wr7 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
          sbwm\shell_based_wireframe_model.sbwm_boundary | (
          'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
          (SIZEOF(QUERY ( eloop <* QUERY ( wsb <* ws\wire_shell.
          wire_shell_extent | ('CONFIG_CONTROL_DESIGN.EDGE_LOOP' IN
          TYPEOF(wsb)) ) | (NOT (SIZEOF(QUERY ( el <* eloop\path.
          edge_list | (NOT (valid_wireframe_vertex_point(el.
          edge_element.edge_start\vertex_point.vertex_geometry) AND
          valid_wireframe_vertex_point(el.edge_element.edge_end\
          vertex_point.vertex_geometry))) )) = 0)) ))
          = 0)) )) = 0)) )) = 0);
wr8 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
          sbwm\shell_based_wireframe_model.sbwm_boundary | (
          'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
          (SIZEOF(QUERY ( vloop <* QUERY ( wsb <* ws\wire_shell.
          wire_shell_extent | ('CONFIG_CONTROL_DESIGN.VERTEX_LOOP' IN
           TYPEOF(wsb)) ) | (NOT (
          'CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(vloop\
          vertex_loop.loop_vertex))) )) = 0)) )) = 0)) )) = 0);
wr9 : (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
          'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
          TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( ws <* QUERY ( sb <*
```

```
                sbwm\shell_based_wireframe_model.sbwm_boundary | (
                'CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(sb)) ) | (NOT
                (SIZEOF(QUERY ( vloop <* QUERY ( wsb <* ws\wire_shell.
                wire_shell_extent | ('CONFIG_CONTROL_DESIGN.VERTEX_LOOP' IN
                 TYPEOF(wsb)) ) | (NOT valid_wireframe_vertex_point(vloop\
                vertex_loop.loop_vertex\vertex_point.vertex_geometry)) )) =
                 0)) )) = 0)) )) = 0);
    wr10: (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
                'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
                TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( vs <* QUERY ( sb <*
                sbwm\shell_based_wireframe_model.sbwm_boundary | (
                'CONFIG_CONTROL_DESIGN.VERTEX_SHELL' IN TYPEOF(sb)) ) | (
                NOT ('CONFIG_CONTROL_DESIGN.VERTEX_POINT' IN TYPEOF(vs\
                vertex_shell.vertex_shell_extent.loop_vertex))) )) = 0)) ))
                = 0);
    wr11: (SIZEOF(QUERY ( sbwm <* QUERY ( it <* SELF.items | (
                'CONFIG_CONTROL_DESIGN.SHELL_BASED_WIREFRAME_MODEL' IN
                TYPEOF(it)) ) | (NOT (SIZEOF(QUERY ( vs <* QUERY ( sb <*
                sbwm\shell_based_wireframe_model.sbwm_boundary | (
                'CONFIG_CONTROL_DESIGN.VERTEX_SHELL' IN TYPEOF(sb)) ) | (
                NOT valid_wireframe_vertex_point(vs\vertex_shell.
                vertex_shell_extent.loop_vertex\vertex_point.
                vertex_geometry)) )) = 0)) )) = 0);
    wr12: (SIZEOF(QUERY ( mi <* QUERY ( it <* SELF.items | (
                'CONFIG_CONTROL_DESIGN.MAPPED_ITEM' IN TYPEOF(it)) ) | (
                NOT (('CONFIG_CONTROL_DESIGN.' +
                'SHELL_BASED_WIREFRAME_SHAPE_REPRESENTATION') IN TYPEOF(mi\
                mapped_item.mapping_source.mapped_representation))) ))
                = 0);
    wr13: (SELF.context_of_items\geometric_representation_context.
                coordinate_space_dimension = 3);
END_ENTITY; -- shell_based_wireframe_shape_representation

ENTITY si_unit
  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; -- si_unit
```

```
ENTITY solid_angle_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.SOLID_ANGLE_UNIT' IN TYPEOF(SELF\
             measure_with_unit.unit_component));
END_ENTITY; -- solid_angle_measure_with_unit

ENTITY solid_angle_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 0) AND (SELF\
             named_unit.dimensions.mass_exponent = 0) AND (SELF\
             named_unit.dimensions.time_exponent = 0) AND (SELF\
             named_unit.dimensions.electric_current_exponent = 0) AND (
             SELF\named_unit.dimensions.
             thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
             .dimensions.amount_of_substance_exponent = 0) AND (SELF\
             named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- solid_angle_unit

ENTITY solid_model
  SUPERTYPE OF (manifold_solid_brep)
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- solid_model

ENTITY specified_higher_usage_occurrence
  SUBTYPE OF (assembly_component_usage);
    upper_usage : assembly_component_usage;
    next_usage  : next_assembly_usage_occurrence;
  UNIQUE
    ur1 : upper_usage, next_usage;
  WHERE
    wr1: (SELF :<>: upper_usage);
    wr2: (SELF\product_definition_relationship.
             relating_product_definition :=: upper_usage.
             relating_product_definition);
    wr3: (SELF\product_definition_relationship.
             related_product_definition :=: next_usage.
             related_product_definition);
    wr4: (upper_usage.related_product_definition :=: next_usage.
             relating_product_definition);
    wr5: (NOT ('CONFIG_CONTROL_DESIGN.PROMISSORY_USAGE_OCCURRENCE' IN
```

```
              TYPEOF(upper_usage)));
END_ENTITY; -- specified_higher_usage_occurrence

ENTITY spherical_surface
  SUBTYPE OF (elementary_surface);
    radius : positive_length_measure;
END_ENTITY; -- spherical_surface

ENTITY start_request
  SUBTYPE OF (action_request_assignment);
    items : SET [1:?] OF start_request_item;
END_ENTITY; -- start_request

ENTITY start_work
  SUBTYPE OF (action_assignment);
    items : SET [1:?] OF work_item;
END_ENTITY; -- start_work

ENTITY supplied_part_relationship
  SUBTYPE OF (product_definition_relationship);
END_ENTITY; -- supplied_part_relationship

ENTITY surface
  SUPERTYPE OF (ONEOF (elementary_surface,swept_surface,bounded_surface,
      offset_surface,surface_replica))
  SUBTYPE OF (geometric_representation_item);
END_ENTITY; -- surface

ENTITY surface_curve
  SUPERTYPE OF (ONEOF (intersection_curve,seam_curve) ANDOR
      bounded_surface_curve)
  SUBTYPE OF (curve);
    curve_3d              : curve;
    associated_geometry   : LIST [1:2] OF pcurve_or_surface;
    master_representation : preferred_surface_curve_representation;
  DERIVE
    basis_surface : SET [1:2] OF surface := get_basis_surface(SELF);
  WHERE
    wr1: (curve_3d.dim = 3);
    wr2: (('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(associated_geometry[
             1])) OR (master_representation <> pcurve_s1));
    wr3: (('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(associated_geometry[
```

```
              2])) OR (master_representation <> pcurve_s2));
    wr4: (NOT ('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(curve_3d)));
END_ENTITY; -- surface_curve

ENTITY surface_of_linear_extrusion
  SUBTYPE OF (swept_surface);
    extrusion_axis : vector;
END_ENTITY; -- surface_of_linear_extrusion

ENTITY surface_of_revolution
  SUBTYPE OF (swept_surface);
    axis_position : axis1_placement;
  DERIVE
    axis_line : line := dummy_gri || curve() || line(axis_position.
                    location,dummy_gri || vector(axis_position.z,1));
END_ENTITY; -- surface_of_revolution

ENTITY surface_patch
  SUBTYPE OF (founded_item);
    parent_surface : bounded_surface;
    u_transition   : transition_code;
    v_transition   : transition_code;
    u_sense        : BOOLEAN;
    v_sense        : BOOLEAN;
  INVERSE
    using_surfaces : BAG [1:?] OF rectangular_composite_surface FOR
                         segments;
  WHERE
    wr1: (NOT ('CONFIG_CONTROL_DESIGN.CURVE_BOUNDED_SURFACE' IN TYPEOF(
              parent_surface)));
END_ENTITY; -- surface_patch

ENTITY surface_replica
  SUBTYPE OF (surface);
    parent_surface : surface;
    transformation : cartesian_transformation_operator_3d;
  WHERE
    wr1: acyclic_surface_replica(SELF,parent_surface);
END_ENTITY; -- surface_replica

ENTITY swept_surface
  SUPERTYPE OF (ONEOF (surface_of_linear_extrusion,
```

```
      surface_of_revolution))
  SUBTYPE OF (surface);
    swept_curve : curve;
END_ENTITY; -- swept_surface

ENTITY topological_representation_item
  SUPERTYPE OF (ONEOF (vertex,edge,face_bound,face,vertex_shell,
      wire_shell,connected_edge_set,connected_face_set,loop ANDOR path))
  SUBTYPE OF (representation_item);
END_ENTITY; -- topological_representation_item

ENTITY toroidal_surface
  SUBTYPE OF (elementary_surface);
    major_radius : positive_length_measure;
    minor_radius : positive_length_measure;
END_ENTITY; -- toroidal_surface

ENTITY trimmed_curve
  SUBTYPE OF (bounded_curve);
    basis_curve           : curve;
    trim_1                : SET [1:2] OF trimming_select;
    trim_2                : SET [1:2] OF trimming_select;
    sense_agreement       : BOOLEAN;
    master_representation : trimming_preference;
  WHERE
    wr1: ((HIINDEX(trim_1) = 1) OR (TYPEOF(trim_1[1]) <>
         TYPEOF(trim_1[2])));
    wr2: ((HIINDEX(trim_2) = 1) OR (TYPEOF(trim_2[1]) <>
         TYPEOF(trim_2[2])));
END_ENTITY; -- trimmed_curve

ENTITY uncertainty_measure_with_unit
  SUBTYPE OF (measure_with_unit);
    name        : label;
    description : text;
  WHERE
    wr1: valid_measure_value(SELF\measure_with_unit.value_component);
END_ENTITY; -- uncertainty_measure_with_unit

ENTITY uniform_curve
  SUBTYPE OF (b_spline_curve);
END_ENTITY; -- uniform_curve
```

```
ENTITY uniform_surface
  SUBTYPE OF (b_spline_surface);
END_ENTITY; -- uniform_surface

ENTITY vector
  SUBTYPE OF (geometric_representation_item);
    orientation : direction;
    magnitude   : length_measure;
  WHERE
    wr1: (magnitude >= 0);
END_ENTITY; -- vector

ENTITY versioned_action_request;
    id          : identifier;
    version     : label;
    purpose     : text;
    description : text;
END_ENTITY; -- versioned_action_request

ENTITY vertex
  SUBTYPE OF (topological_representation_item);
END_ENTITY; -- vertex

ENTITY vertex_loop
  SUBTYPE OF (loop);
    loop_vertex : vertex;
END_ENTITY; -- vertex_loop

ENTITY vertex_point
  SUBTYPE OF (vertex, geometric_representation_item);
    vertex_geometry : point;
END_ENTITY; -- vertex_point

ENTITY vertex_shell
  SUBTYPE OF (topological_representation_item);
    vertex_shell_extent : vertex_loop;
END_ENTITY; -- vertex_shell

ENTITY volume_measure_with_unit
  SUBTYPE OF (measure_with_unit);
  WHERE
    wr1: ('CONFIG_CONTROL_DESIGN.VOLUME_UNIT' IN TYPEOF(SELF\
```

```
              measure_with_unit.unit_component));
END_ENTITY; -- volume_measure_with_unit

ENTITY volume_unit
  SUBTYPE OF (named_unit);
  WHERE
    wr1: ((SELF\named_unit.dimensions.length_exponent = 3) AND (SELF\
              named_unit.dimensions.mass_exponent = 0) AND (SELF\
              named_unit.dimensions.time_exponent = 0) AND (SELF\
              named_unit.dimensions.electric_current_exponent = 0) AND (
              SELF\named_unit.dimensions.
              thermodynamic_temperature_exponent = 0) AND (SELF\named_unit
              .dimensions.amount_of_substance_exponent = 0) AND (SELF\
              named_unit.dimensions.luminous_intensity_exponent = 0));
END_ENTITY; -- volume_unit

ENTITY week_of_year_and_day_date
  SUBTYPE OF (date);
    week_component : week_in_year_number;
    day_component  : OPTIONAL day_in_week_number;
END_ENTITY; -- week_of_year_and_day_date

ENTITY wire_shell
  SUBTYPE OF (topological_representation_item);
    wire_shell_extent : SET [1:?] OF loop;
  WHERE
    wr1: (NOT mixed_loop_type_set(wire_shell_extent));
END_ENTITY; -- wire_shell

RULE acu_requires_security_classification FOR (assembly_component_usage,
              cc_design_security_classification);
WHERE
  wr1: (SIZEOF(QUERY ( acu <* assembly_component_usage | (NOT (SIZEOF(
            QUERY ( ccdsc <* cc_design_security_classification | (acu IN
            ccdsc.items) )) = 1)) )) = 0);
END_RULE; -- acu_requires_security_classification

RULE application_context_requires_ap_definition FOR (application_context,
            application_protocol_definition);
WHERE
  wr1: (SIZEOF(QUERY ( ac <* application_context | (NOT (SIZEOF(
            QUERY ( apd <* application_protocol_definition | ((ac :=: apd.
```

```
            application) AND (apd.
            application_interpreted_model_schema_name =
            'config_control_design')) )) = 1)) )) = 0);
END_RULE; -- application_context_requires_ap_definition

RULE approval_date_time_constraints FOR (approval_date_time);
WHERE
  wr1: (SIZEOF(QUERY ( adt <* approval_date_time | (NOT (SIZEOF(TYPEOF(
            adt.date_time) * ['CONFIG_CONTROL_DESIGN.DATE_AND_TIME'])
            = 1)) )) = 0);
END_RULE; -- approval_date_time_constraints

RULE approval_person_organization_constraints FOR (
            approval_person_organization);
WHERE
  wr1: (SIZEOF(QUERY ( apo <* approval_person_organization | (NOT (
            SIZEOF(TYPEOF(apo.person_organization) * [
            'CONFIG_CONTROL_DESIGN.PERSON_AND_ORGANIZATION'])
            = 1)) )) = 0);
END_RULE; -- approval_person_organization_constraints

RULE approval_requires_approval_date_time FOR (approval,
            approval_date_time);
WHERE
  wr1: (SIZEOF(QUERY ( app <* approval | (NOT (SIZEOF(QUERY ( adt <*
            approval_date_time | (app :=: adt.dated_approval) )) = 1)) ))
            = 0);
END_RULE; -- approval_requires_approval_date_time

RULE approval_requires_approval_person_organization FOR (approval,
            approval_person_organization);
WHERE
  wr1: (SIZEOF(QUERY ( app <* approval | (NOT (SIZEOF(QUERY ( apo <*
            approval_person_organization | (app :=: apo.
            authorized_approval) )) >= 1)) )) = 0);
END_RULE; -- approval_requires_approval_person_organization

RULE approvals_are_assigned FOR (approval, approval_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( app <* approval | (NOT (SIZEOF(QUERY ( aa <*
            approval_assignment | (app :=: aa.assigned_approval) ))
```

```
            >= 1)) )) = 0);
END_RULE; -- approvals_are_assigned

RULE as_required_quantity FOR (measure_with_unit);
WHERE
  wr1: (SIZEOF(QUERY ( m <* measure_with_unit | ((
            'CONFIG_CONTROL_DESIGN.DESCRIPTIVE_MEASURE' IN TYPEOF(m.
            value_component)) AND (NOT (m.value_component =
            'as_required'))) )) = 0);
END_RULE; -- as_required_quantity

RULE certification_requires_approval FOR (certification,
            cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( cert <* certification | (NOT (SIZEOF(
            QUERY ( ccda <* cc_design_approval | (cert IN ccda.items) )) =
             1)) )) = 0);
END_RULE; -- certification_requires_approval

RULE certification_requires_date_time FOR (certification,
            cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( cert <* certification | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | (cert IN
             ccdta.items) )) = 1)) )) = 0);
END_RULE; -- certification_requires_date_time

RULE change_request_requires_approval FOR (change_request,
            cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( cr <* change_request | (NOT (SIZEOF(
            QUERY ( ccda <* cc_design_approval | (cr IN ccda.items) ))
            = 1)) )) = 0);
END_RULE; -- change_request_requires_approval

RULE change_request_requires_date_time FOR (change_request,
            cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( cr <* change_request | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | (cr IN
            ccdta.items) )) = 1)) )) = 0);
END_RULE; -- change_request_requires_date_time
```

```
RULE change_request_requires_person_organization FOR (change_request,
           cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( cr <* change_request | (NOT (SIZEOF(
           QUERY ( ccpoa <* cc_design_person_and_organization_assignment
            | (cr IN ccpoa.items) )) >= 1)) )) = 0);
END_RULE; -- change_request_requires_person_organization

RULE change_requires_approval FOR (change, cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( chg <* change | (NOT (SIZEOF(QUERY ( ccda <*
           cc_design_approval | (chg IN ccda.items) )) = 1)) )) = 0);
END_RULE; -- change_requires_approval

RULE change_requires_date_time FOR (change,
           cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( chg <* change | (NOT (SIZEOF(QUERY ( ccdta <*
           cc_design_date_and_time_assignment | ((chg IN ccdta.items) AND
            (ccdta.role.name = 'start_date')) )) = 1)) )) = 0);
END_RULE; -- change_requires_date_time

RULE compatible_dimension FOR (cartesian_point, direction,
           representation_context, geometric_representation_context);
WHERE
  wr1: (SIZEOF(QUERY ( x <* cartesian_point | (SIZEOF(QUERY ( y <*
           geometric_representation_context | (item_in_context(x,y) AND (
           HIINDEX(x.coordinates) <> y.coordinate_space_dimension)) )) >
           0) )) = 0);
  wr2: (SIZEOF(QUERY ( x <* direction | (SIZEOF(QUERY ( y <*
           geometric_representation_context | (item_in_context(x,y) AND (
           HIINDEX(x.direction_ratios) <> y.coordinate_space_dimension))
           )) > 0) )) = 0);
END_RULE; -- compatible_dimension

RULE configuration_item_requires_approval FOR (configuration_item,
           cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( ci <* configuration_item | (NOT (SIZEOF(
           QUERY ( ccda <* cc_design_approval | (ci IN ccda.items) ))
           = 1)) )) = 0);
END_RULE; -- configuration_item_requires_approval
```

```
RULE configuration_item_requires_person_organization FOR (
           configuration_item,
           cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( ci <* configuration_item | (NOT (SIZEOF(
           QUERY ( ccdpoa <* cc_design_person_and_organization_assignment
            | (ci IN ccdpoa.items) )) = 1)) )) = 0);
END_RULE; -- configuration_item_requires_person_organization

RULE contract_requires_approval FOR (contract, cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( c <* contract | (NOT (SIZEOF(QUERY ( ccda <*
           cc_design_approval | (c IN ccda.items) )) = 1)) )) = 0);
END_RULE; -- contract_requires_approval

RULE contract_requires_person_organization FOR (contract,
           cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( c <* contract | (NOT (SIZEOF(QUERY ( ccdpoa <*
           cc_design_person_and_organization_assignment | (c IN ccdpoa.
           items) )) = 1)) )) = 0);
END_RULE; -- contract_requires_person_organization

RULE coordinated_assembly_and_shape FOR (next_assembly_usage_occurrence);
WHERE
  wr1: (SIZEOF(QUERY ( nauo <* next_assembly_usage_occurrence | (NOT
           assembly_shape_is_defined(nauo,'CONFIG_CONTROL_DESIGN')) )) =
           0);
END_RULE; -- coordinated_assembly_and_shape

RULE dependent_instantiable_action_directive FOR (action_directive);
WHERE
  wr1: (SIZEOF(QUERY ( ad <* action_directive | (NOT (SIZEOF(USEDIN(ad,
           '')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_action_directive

RULE dependent_instantiable_approval_status FOR (approval_status);
WHERE
  wr1: (SIZEOF(QUERY ( ast <* approval_status | (NOT (SIZEOF(USEDIN(ast,
           '')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_approval_status
```

```
RULE dependent_instantiable_certification_type FOR (certification_type);
WHERE
  wr1: (SIZEOF(QUERY ( ct <* certification_type | (NOT (SIZEOF(USEDIN(ct,
           '')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_certification_type

RULE dependent_instantiable_contract_type FOR (contract_type);
WHERE
  wr1: (SIZEOF(QUERY ( ct <* contract_type | (NOT (SIZEOF(USEDIN(ct,''))
           >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_contract_type

RULE dependent_instantiable_date FOR (date);
WHERE
  wr1: (SIZEOF(QUERY ( dt <* date | (NOT (SIZEOF(USEDIN(dt,'')) >= 1)) ))
           = 0);
END_RULE; -- dependent_instantiable_date

RULE dependent_instantiable_date_time_role FOR (date_time_role);
WHERE
  wr1: (SIZEOF(QUERY ( dtr <* date_time_role | (NOT (SIZEOF(USEDIN(dtr,
           '')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_date_time_role

RULE dependent_instantiable_document_type FOR (document_type);
WHERE
  wr1: (SIZEOF(QUERY ( dt <* document_type | (NOT (SIZEOF(USEDIN(dt,''))
           >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_document_type

RULE dependent_instantiable_named_unit FOR (named_unit);
WHERE
  wr1: (SIZEOF(QUERY ( nu <* named_unit | (NOT (SIZEOF(USEDIN(nu,'')) >=
            1)) )) = 0);
END_RULE; -- dependent_instantiable_named_unit

RULE dependent_instantiable_parametric_representation_context FOR (
           parametric_representation_context);
WHERE
  wr1: (SIZEOF(QUERY ( prc <* parametric_representation_context | (NOT (
           SIZEOF(USEDIN(prc,'')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_parametric_representation_context
```

```
RULE dependent_instantiable_person_and_organization_role FOR (
           person_and_organization_role);
WHERE
  wr1: (SIZEOF(QUERY ( poar <* person_and_organization_role | (NOT (
           SIZEOF(USEDIN(poar,'')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_person_and_organization_role

RULE dependent_instantiable_representation_item FOR
          (representation_item);
WHERE
  wr1: (SIZEOF(QUERY ( ri <* representation_item | (NOT (SIZEOF(USEDIN(
           ri,'')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_representation_item

RULE dependent_instantiable_security_classification_level FOR (
           security_classification_level);
WHERE
  wr1: (SIZEOF(QUERY ( scl <* security_classification_level | (NOT (
           SIZEOF(USEDIN(scl,'')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_security_classification_level

RULE dependent_instantiable_shape_representation FOR (
           shape_representation);
WHERE
  wr1: (SIZEOF(QUERY ( sr <* shape_representation | (NOT (SIZEOF(USEDIN(
           sr,'')) >= 1)) )) = 0);
END_RULE; -- dependent_instantiable_shape_representation

RULE design_context_for_property FOR (product_definition);
WHERE
  wr1: (SIZEOF(QUERY ( pd <* product_definition | ((SIZEOF(USEDIN(pd,
           'CONFIG_CONTROL_DESIGN.' + 'PROPERTY_DEFINITION.DEFINITION') +
            QUERY ( pdr <* USEDIN(pd,'CONFIG_CONTROL_DESIGN.' +
           'PRODUCT_DEFINITION_RELATIONSHIP.RELATED_PRODUCT_DEFINITION')
            | (SIZEOF(USEDIN(pdr,
           'CONFIG_CONTROL_DESIGN.PROPERTY_DEFINITION.' + 'DEFINITION'))
           >= 1) )) >= 1) AND (NOT (
           'CONFIG_CONTROL_DESIGN.DESIGN_CONTEXT' IN TYPEOF(pd.
           frame_of_reference)))) )) = 0);
END_RULE; -- design_context_for_property
```

```
RULE document_to_product_definition FOR (
           cc_design_specification_reference);
WHERE
  wr1: (SIZEOF(QUERY ( sp <* cc_design_specification_reference | (NOT ((
           (('CONFIG_CONTROL_DESIGN.DOCUMENT_RELATIONSHIP.' +
           'RELATING_DOCUMENT') IN ROLESOF(sp\document_reference.
           assigned_document)) AND (SIZEOF(QUERY ( it <* sp.items | (NOT
           ('CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION' IN TYPEOF(it))) ))
           = 0)) OR (NOT (('CONFIG_CONTROL_DESIGN.DOCUMENT_RELATIONSHIP.'
           + 'RELATING_DOCUMENT') IN ROLESOF(sp\document_reference.
           assigned_document))))) )) = 0);
END_RULE; -- document_to_product_definition

RULE effectivity_requires_approval FOR (effectivity, cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( eff <* effectivity | (NOT (SIZEOF(
           QUERY ( ccda <* cc_design_approval | (eff IN ccda.items) )) =
           1)) )) = 0);
END_RULE; -- effectivity_requires_approval

RULE geometric_representation_item_3d FOR
          (geometric_representation_item);
WHERE
  wr1: (SIZEOF(QUERY ( gri <* geometric_representation_item | (NOT ((
           dimension_of(gri) = 3) OR (SIZEOF(QUERY ( ur <*
           using_representations(gri) | (
           'CONFIG_CONTROL_DESIGN.DEFINITIONAL_REPRESENTATION' IN TYPEOF(
           ur)) )) > 0))) )) = 0);
END_RULE; -- geometric_representation_item_3d

RULE global_unit_assignment FOR (global_unit_assigned_context);
WHERE
  wr1: (SIZEOF(QUERY ( guac <* global_unit_assigned_context | (NOT (
           SIZEOF(guac.units) = 3)) )) = 0);
  wr2: (SIZEOF(QUERY ( guac <* global_unit_assigned_context | (NOT ((
           SIZEOF(QUERY ( u <* guac.units | (
           'CONFIG_CONTROL_DESIGN.LENGTH_UNIT' IN TYPEOF(u)) )) = 1) AND
           (SIZEOF(QUERY ( u <* guac.units | (
           'CONFIG_CONTROL_DESIGN.PLANE_ANGLE_UNIT' IN TYPEOF(u)) )) = 1)
           AND (SIZEOF(QUERY ( u <* guac.units | (
           'CONFIG_CONTROL_DESIGN.SOLID_ANGLE_UNIT' IN TYPEOF(u)) ))
```

```
            = 1))) )) = 0);
END_RULE; -- global_unit_assignment

RULE no_shape_for_make_from FOR (design_make_from_relationship);
WHERE
  wr1: (SIZEOF(QUERY ( dmfr <* design_make_from_relationship | (NOT (
            SIZEOF(QUERY ( pd <* USEDIN(dmfr,'CONFIG_CONTROL_DESIGN.' +
            'PROPERTY_DEFINITION.DEFINITION') | (
            'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_SHAPE'
            IN TYPEOF(pd)) )) = 0)) )) = 0);
END_RULE; -- no_shape_for_make_from

RULE no_shape_for_supplied_part FOR (supplied_part_relationship);
WHERE
  wr1: (SIZEOF(QUERY ( spr <* supplied_part_relationship | (NOT (SIZEOF(
            QUERY ( pd <* USEDIN(spr,'CONFIG_CONTROL_DESIGN.' +
            'PROPERTY_DEFINITION.DEFINITION') | (
            'CONFIG_CONTROL_DESIGN.PRODUCT_DEFINITION_SHAPE'
            IN TYPEOF(pd)) )) = 0)) )) = 0);
END_RULE; -- no_shape_for_supplied_part

RULE product_concept_requires_configuration_item FOR (product_concept,
            configuration_item);
WHERE
  wr1: (SIZEOF(QUERY ( pc <* product_concept | (NOT (SIZEOF(
            QUERY ( ci <* configuration_item | (pc :=: ci.item_concept) ))
            >= 1)) )) = 0);
END_RULE; -- product_concept_requires_configuration_item

RULE product_definition_requires_approval FOR (product_definition,
            cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( pd <* product_definition | (NOT (SIZEOF(
            QUERY ( ccda <* cc_design_approval | (pd IN ccda.items) ))
            = 1)) )) = 0);
END_RULE; -- product_definition_requires_approval

RULE product_definition_requires_date_time FOR (product_definition,
            cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( pd <* product_definition | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | (pd IN
```

```
           ccdta.items) )) = 1)) )) = 0);
END_RULE; -- product_definition_requires_date_time

RULE product_definition_requires_person_organization FOR (
           product_definition,
           cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( pd <* product_definition | (NOT (SIZEOF(
           QUERY ( ccdpoa <* cc_design_person_and_organization_assignment
            | (pd IN ccdpoa.items) )) = 1)) )) = 0);
END_RULE; -- product_definition_requires_person_organization

RULE product_requires_person_organization FOR (product,
           cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( prod <* product | (NOT (SIZEOF(QUERY ( ccdpoa <*
           cc_design_person_and_organization_assignment | (prod IN ccdpoa
           .items) )) = 1)) )) = 0);
END_RULE; -- product_requires_person_organization

RULE product_requires_product_category FOR (product,
           product_related_product_category);
WHERE
  wr1: (SIZEOF(QUERY ( prod <* product | (NOT (SIZEOF(QUERY ( prpc <*
           product_related_product_category | ((prod IN prpc.products)
           AND (prpc.name IN ['assembly','inseparable_assembly','detail',
           'customer_furnished_equipment'])) )) = 1)) )) = 0);
END_RULE; -- product_requires_product_category

RULE product_requires_version FOR (product,
           product_definition_formation);
WHERE
  wr1: (SIZEOF(QUERY ( prod <* product | (NOT (SIZEOF(QUERY ( pdf <*
           product_definition_formation | (prod :=: pdf.of_product) )) >=
            1)) )) = 0);
END_RULE; -- product_requires_version

RULE product_version_requires_approval FOR (product_definition_formation,
           cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( pdf <* product_definition_formation | (NOT (
           SIZEOF(QUERY ( ccda <* cc_design_approval |
```

```
            (pdf IN ccda.items) )) = 1)) )) = 0);
END_RULE; -- product_version_requires_approval

RULE product_version_requires_person_organization FOR (
            product_definition_formation,
            cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( pdf <* product_definition_formation | (NOT (
            SIZEOF(QUERY ( ccdpoa <*
            cc_design_person_and_organization_assignment | ((pdf IN ccdpoa
            .items) AND (ccdpoa.role.name = 'creator')) )) = 1)) )) = 0);
  wr2: (SIZEOF(QUERY ( pdf <* product_definition_formation | (NOT (
            SIZEOF(QUERY ( ccdpoa <*
            cc_design_person_and_organization_assignment | ((pdf IN ccdpoa
            .items) AND (ccdpoa.role.name IN ['design_supplier',
            'part_supplier'])) )) >= 1)) )) = 0);
END_RULE; -- product_version_requires_person_organization

RULE product_version_requires_security_classification FOR (
            product_definition_formation,
            cc_design_security_classification);
WHERE
  wr1: (SIZEOF(QUERY ( pdf <* product_definition_formation | (NOT (
            SIZEOF(QUERY ( ccdsc <* cc_design_security_classification | (
            pdf IN ccdsc.items) )) = 1)) )) = 0);
END_RULE; -- product_version_requires_security_classification

RULE restrict_action_request_status FOR (action_request_status);
WHERE
  wr1: (SIZEOF(QUERY ( ars <* action_request_status | (NOT (ars.status
            IN ['proposed','in_work','issued','hold'])) )) = 0);
END_RULE; -- restrict_action_request_status

RULE restrict_approval_status FOR (approval_status);
WHERE
  wr1: (SIZEOF(QUERY ( ast <* approval_status | (NOT (ast.name IN [
            'approved','not_yet_approved','disapproved','withdrawn'])) ))
            = 0);
END_RULE; -- restrict_approval_status

RULE restrict_certification_type FOR (certification_type);
WHERE
```

```
  wr1: (SIZEOF(QUERY ( ct <* certification_type | (NOT (ct.description
           IN ['design_supplier','part_supplier'])) )) = 0);
END_RULE; -- restrict_certification_type

RULE restrict_contract_type FOR (contract_type);
WHERE
  wr1: (SIZEOF(QUERY ( ct <* contract_type | (NOT (ct.description IN [
           'fixed_price','cost_plus'])) )) = 0);
END_RULE; -- restrict_contract_type

RULE restrict_date_time_role FOR (date_time_role);
WHERE
  wr1: (SIZEOF(QUERY ( dtr <* date_time_role | (NOT (dtr.name IN [
           'creation_date','request_date','release_date','start_date',
           'contract_date','certification_date','sign_off_date',
           'classification_date','declassification_date'])) )) = 0);
END_RULE; -- restrict_date_time_role

RULE restrict_document_type FOR (document_type);
WHERE
  wr1: (SIZEOF(QUERY ( dt <* document_type | (NOT (dt.product_data_type
           IN ['material_specification','process_specification',
           'design_specification','surface_finish_specification',
           'cad_filename','drawing'])) )) = 0);
END_RULE; -- restrict_document_type

RULE restrict_person_organization_role FOR
          (person_and_organization_role);
WHERE
  wr1: (SIZEOF(QUERY ( por <* person_and_organization_role | (NOT (por.
           name IN ['request_recipient','initiator','part_supplier',
           'design_supplier','configuration_manager','contractor',
           'classification_officer','creator','design_owner'])) )) = 0);
END_RULE; -- restrict_person_organization_role

RULE restrict_product_category_value FOR (
           product_related_product_category);
WHERE
  wr1: (SIZEOF(QUERY ( prpc <* product_related_product_category | (NOT (
           prpc.name IN ['assembly','detail',
           'customer_furnished_equipment','inseparable_assembly','cast',
           'coined','drawn','extruded','forged','formed','machined',
```

```
            'molded','rolled','sheared'])) )) = 0);
END_RULE; -- restrict_product_category_value

RULE restrict_security_classification_level FOR (
            security_classification_level);
WHERE
  wr1: (SIZEOF(QUERY ( scl <* security_classification_level | (NOT (scl.
            name IN ['unclassified','classified','proprietary',
            'confidential','secret','top_secret'])) )) = 0);
END_RULE; -- restrict_security_classification_level

RULE security_classification_optional_date_time FOR (
            security_classification, cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sc <* security_classification | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | ((sc IN
            ccdta.items) AND ('declassification_date' = ccdta.role.name))
            )) <= 1)) )) = 0);
END_RULE; -- security_classification_optional_date_time

RULE security_classification_requires_approval FOR (
            security_classification, cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( sc <* security_classification | (NOT (SIZEOF(
            QUERY ( ccda <* cc_design_approval | (sc IN ccda.items) ))
            = 1)) )) = 0);
END_RULE; -- security_classification_requires_approval

RULE security_classification_requires_date_time FOR (
            security_classification, cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sc <* security_classification | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | ((sc IN
            ccdta.items) AND ('classification_date' = ccdta.role.name)) ))
            = 1)) )) = 0);
END_RULE; -- security_classification_requires_date_time

RULE security_classification_requires_person_organization FOR (
            security_classification,
            cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sc <* security_classification | (NOT (SIZEOF(
```

```
            QUERY ( ccdpoa <* cc_design_person_and_organization_assignment
             | (sc IN ccdpoa.items) )) = 1)) )) = 0);
END_RULE; -- security_classification_requires_person_organization

RULE start_request_requires_approval FOR (start_request,
            cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( sr <* start_request | (NOT (SIZEOF(
            QUERY ( ccda <* cc_design_approval | (sr IN ccda.items) ))
            = 1)) )) = 0);
END_RULE; -- start_request_requires_approval

RULE start_request_requires_date_time FOR (start_request,
            cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sr <* start_request | (NOT (SIZEOF(
            QUERY ( ccdta <* cc_design_date_and_time_assignment | (sr IN
            ccdta.items) )) = 1)) )) = 0);
END_RULE; -- start_request_requires_date_time

RULE start_request_requires_person_organization FOR (start_request,
            cc_design_person_and_organization_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sr <* start_request | (NOT (SIZEOF(
            QUERY ( ccdpoa <* cc_design_person_and_organization_assignment
             | (sr IN ccdpoa.items) )) >= 1)) )) = 0);
END_RULE; -- start_request_requires_person_organization

RULE start_work_requires_approval FOR (start_work, cc_design_approval);
WHERE
  wr1: (SIZEOF(QUERY ( sw <* start_work | (NOT (SIZEOF(QUERY ( ccda <*
            cc_design_approval | (sw IN ccda.items) )) = 1)) )) = 0);
END_RULE; -- start_work_requires_approval

RULE start_work_requires_date_time FOR (start_work,
            cc_design_date_and_time_assignment);
WHERE
  wr1: (SIZEOF(QUERY ( sw <* start_work | (NOT (SIZEOF(QUERY ( ccdta <*
            cc_design_date_and_time_assignment | ((sw IN ccdta.items) AND
            (ccdta.role.name = 'start_date')) )) = 1)) )) = 0);
END_RULE; -- start_work_requires_date_time
```

```
RULE subtype_mandatory_action FOR (action);
WHERE
  wr1: (SIZEOF(QUERY ( act <* action | (NOT (
           'CONFIG_CONTROL_DESIGN.DIRECTED_ACTION' IN TYPEOF(act))) )) =
           0);
END_RULE; -- subtype_mandatory_action

RULE subtype_mandatory_effectivity FOR (effectivity);
WHERE
  wr1: (SIZEOF(QUERY ( eff <* effectivity | (NOT ((SIZEOF([
           'CONFIG_CONTROL_DESIGN.SERIAL_NUMBERED_EFFECTIVITY',
           'CONFIG_CONTROL_DESIGN.LOT_EFFECTIVITY',
           'CONFIG_CONTROL_DESIGN.DATED_EFFECTIVITY'] * TYPEOF(eff)) = 1)
           AND ('CONFIG_CONTROL_DESIGN.CONFIGURATION_EFFECTIVITY' IN
           TYPEOF(eff)))) )) = 0);
END_RULE; -- subtype_mandatory_effectivity

RULE subtype_mandatory_product_context FOR (product_context);
WHERE
  wr1: (SIZEOF(QUERY ( pc <* product_context | (NOT (
           'CONFIG_CONTROL_DESIGN.MECHANICAL_CONTEXT' IN TYPEOF(pc))) ))
           = 0);
END_RULE; -- subtype_mandatory_product_context

RULE subtype_mandatory_product_definition_formation FOR (
           product_definition_formation);
WHERE
  wr1: (SIZEOF(QUERY ( pdf <* product_definition_formation | (NOT ((
           'CONFIG_CONTROL_DESIGN.' +
           'PRODUCT_DEFINITION_FORMATION_WITH_SPECIFIED_SOURCE') IN
           TYPEOF(pdf))) )) = 0);
END_RULE; -- subtype_mandatory_product_definition_formation

RULE subtype_mandatory_product_definition_usage FOR (
           product_definition_usage);
WHERE
  wr1: (SIZEOF(QUERY ( pdu <* product_definition_usage | (NOT ((
           'CONFIG_CONTROL_DESIGN.' + 'ASSEMBLY_COMPONENT_USAGE') IN
           TYPEOF(pdu))) )) = 0);
END_RULE; -- subtype_mandatory_product_definition_usage
```

```
RULE subtype_mandatory_representation FOR (representation);
WHERE
  wr1: (SIZEOF(QUERY ( rep <* representation | (NOT (
            'CONFIG_CONTROL_DESIGN.SHAPE_REPRESENTATION' IN TYPEOF(rep)))
            )) = 0);
END_RULE; -- subtype_mandatory_representation

RULE subtype_mandatory_representation_context FOR (
            representation_context);
WHERE
  wr1: (SIZEOF(QUERY ( rep_cntxt <* representation_context | (NOT (
            'CONFIG_CONTROL_DESIGN.GEOMETRIC_REPRESENTATION_CONTEXT' IN
            TYPEOF(rep_cntxt))) )) = 0);
END_RULE; -- subtype_mandatory_representation_context

RULE subtype_mandatory_shape_representation FOR (shape_representation);
WHERE
  wr1: (SIZEOF(QUERY ( sr <* shape_representation | (NOT ((SIZEOF([
            'CONFIG_CONTROL_DESIGN.' +
            'ADVANCED_BREP_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.FACETED_BREP_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.MANIFOLD_SURFACE_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.' +
            'EDGE_BASED_WIREFRAME_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.' +
            'SHELL_BASED_WIREFRAME_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.' +
            'GEOMETRICALLY_BOUNDED_SURFACE_SHAPE_REPRESENTATION',
            'CONFIG_CONTROL_DESIGN.' +
            'GEOMETRICALLY_BOUNDED_WIREFRAME_SHAPE_REPRESENTATION'] *
            TYPEOF(sr)) = 1) OR (SIZEOF(QUERY ( it <* sr\representation.
            items | (NOT ('CONFIG_CONTROL_DESIGN.AXIS2_PLACEMENT_3D' IN
            TYPEOF(it))) )) = 0) OR (SIZEOF(QUERY ( sdr <* QUERY ( pdr <*
            USEDIN(sr,
            'CONFIG_CONTROL_DESIGN.PROPERTY_DEFINITION_REPRESENTATION.' +
            'USED_REPRESENTATION') | (
            'CONFIG_CONTROL_DESIGN.SHAPE_DEFINITION_REPRESENTATION' IN
            TYPEOF(pdr)) ) | (NOT (SIZEOF([
            'CONFIG_CONTROL_DESIGN.SHAPE_ASPECT',
            'CONFIG_CONTROL_DESIGN.SHAPE_ASPECT_RELATIONSHIP'] * TYPEOF(
            sdr.definition.definition)) = 1)) )) = 0))) )) = 0);
END_RULE; -- subtype_mandatory_shape_representation
```

```
RULE unique_version_change_order_rule FOR (change);
WHERE
  wr1: (SIZEOF(QUERY ( c <* change | (NOT unique_version_change_order(c.
           assigned_action)) )) = 0);
END_RULE; -- unique_version_change_order_rule

RULE versioned_action_request_requires_solution FOR (
           versioned_action_request, action_request_solution);
WHERE
  wr1: (SIZEOF(QUERY ( ar <* versioned_action_request | (NOT (SIZEOF(
           QUERY ( ars <* action_request_solution | (ar :=: ars.request)
           )) >= 1)) )) = 0);
END_RULE; -- versioned_action_request_requires_solution

RULE versioned_action_request_requires_status FOR (
           versioned_action_request, action_request_status);
WHERE
  wr1: (SIZEOF(QUERY ( ar <* versioned_action_request | (NOT (SIZEOF(
           QUERY ( ars <* action_request_status | (ar :=: ars.
           assigned_request) )) = 1)) )) = 0);
END_RULE; -- versioned_action_request_requires_status

FUNCTION acyclic_curve_replica(rep: curve_replica;
             parent: curve): BOOLEAN;
  IF NOT ('CONFIG_CONTROL_DESIGN.CURVE_REPLICA' IN TYPEOF(parent)) THEN
    RETURN(TRUE);
  END_IF;
  IF parent :=: rep THEN RETURN(FALSE);
  ELSE
    RETURN(acyclic_curve_replica(rep,parent\curve_replica.parent_curve));
  END_IF;
END_FUNCTION; -- acyclic_curve_replica

FUNCTION acyclic_mapped_representation(parent_set: SET OF representation;
             children_set: SET OF representation_item): BOOLEAN;
  LOCAL
    i : INTEGER;
    x : SET OF representation_item;
    y : SET OF representation_item;
  END_LOCAL;
  x := QUERY ( z <* children_set | ('CONFIG_CONTROL_DESIGN.MAPPED_ITEM'
      IN TYPEOF(z)) );
```

```
    IF SIZEOF(x) > 0 THEN
      REPEAT i := 1 TO HIINDEX(x) BY 1;
        IF x[i]\mapped_item.mapping_source.mapped_representation IN
            parent_set THEN RETURN(FALSE);
        END_IF;
        IF NOT acyclic_mapped_representation(parent_set + x[i]\mapped_item
            .mapping_source.mapped_representation,x[i]\mapped_item.
            mapping_source.mapped_representation.items) THEN
          RETURN(FALSE);
        END_IF;
      END_REPEAT;
    END_IF;
    x := children_set - x;
    IF SIZEOF(x) > 0 THEN
      REPEAT i := 1 TO HIINDEX(x) BY 1;
        y := QUERY ( z <* bag_to_set(USEDIN(x[i],'')) | (
            'CONFIG_CONTROL_DESIGN.REPRESENTATION_ITEM' IN TYPEOF(z)) );
        IF NOT acyclic_mapped_representation(parent_set,y) THEN
          RETURN(FALSE);
        END_IF;
      END_REPEAT;
    END_IF;
    RETURN(TRUE);
  END_FUNCTION; -- acyclic_mapped_representation

  FUNCTION acyclic_point_replica(rep: point_replica;
               parent: point): BOOLEAN;
    IF NOT ('CONFIG_CONTROL_DESIGN.POINT_REPLICA' IN TYPEOF(parent)) THEN
      RETURN(TRUE);
    END_IF;
    IF parent :=: rep THEN RETURN(FALSE);
    ELSE
      RETURN(acyclic_point_replica(rep,parent\point_replica.parent_pt));
    END_IF;
  END_FUNCTION; -- acyclic_point_replica

  FUNCTION acyclic_product_category_relationship(
               relation: product_category_relationship;
               children: SET OF product_category): LOGICAL;
    LOCAL
      i               : INTEGER;
      x               : SET OF product_category_relationship;
```

```
    local_children : SET OF product_category;
  END_LOCAL;
  REPEAT i := 1 TO HIINDEX(children) BY 1;
    IF relation.category :=: children[i] THEN RETURN(FALSE);
    END_IF;
  END_REPEAT;
  x := bag_to_set(USEDIN(relation.category,'CONFIG_CONTROL_DESIGN.' +
      'PRODUCT_CATEGORY_RELATIONSHIP.SUB_CATEGORY'));
  local_children := children + relation.category;
  IF SIZEOF(x) > 0 THEN
    REPEAT i := 1 TO HIINDEX(x) BY 1;
      IF NOT acyclic_product_category_relationship(x[i],local_children)
           THEN RETURN(FALSE);
      END_IF;
    END_REPEAT;
  END_IF;
  RETURN(TRUE);
END_FUNCTION; -- acyclic_product_category_relationship

FUNCTION acyclic_product_definition_relationship(
             relation: product_definition_relationship;
             relatives: SET [1:?] OF product_definition;
             specific_relation: STRING): LOGICAL;
  LOCAL
    x : SET OF product_definition_relationship;
  END_LOCAL;
  IF relation.relating_product_definition IN relatives THEN
    RETURN(FALSE);
  END_IF;
  x := QUERY ( pd <* bag_to_set(USEDIN(relation.
      relating_product_definition,'CONFIG_CONTROL_DESIGN.' +
      'PRODUCT_DEFINITION_RELATIONSHIP.' + 'RELATED_PRODUCT_DEFINITION'))
       | (specific_relation IN TYPEOF(pd)) );
  REPEAT i := 1 TO HIINDEX(x) BY 1;
    IF NOT acyclic_product_definition_relationship(x[i],relatives +
        relation.relating_product_definition,specific_relation) THEN
      RETURN(FALSE);
    END_IF;
  END_REPEAT;
  RETURN(TRUE);
END_FUNCTION; -- acyclic_product_definition_relationship
```

```
FUNCTION acyclic_surface_replica(rep: surface_replica;
             parent: surface): BOOLEAN;
  IF NOT ('CONFIG_CONTROL_DESIGN.SURFACE_REPLICA' IN TYPEOF(parent))
       THEN
    RETURN(TRUE);
  END_IF;
  IF parent :=: rep THEN RETURN(FALSE);
  ELSE
    RETURN(acyclic_surface_replica(rep,parent\surface_replica.
        parent_surface));
  END_IF;
END_FUNCTION; -- acyclic_surface_replica

FUNCTION assembly_shape_is_defined(assy: next_assembly_usage_occurrence;
             schma: STRING): BOOLEAN;
  LOCAL
    srr_set   : SET OF shape_representation_relationship := [];
    i         : INTEGER;
    j         : INTEGER;
    sdr_set   : SET OF shape_definition_representation := [];
    pr1_set   : SET OF property_definition := [];
    pdrel_set : SET OF product_definition_relationship := [];
    pr2_set   : SET OF property_definition := [];
  END_LOCAL;
  pr1_set := bag_to_set(USEDIN(assy.related_product_definition,schma +
      '.PROPERTY_DEFINITION.DEFINITION'));
  REPEAT i := 1 TO HIINDEX(pr1_set) BY 1;
    sdr_set := sdr_set + QUERY ( pdr <* USEDIN(pr1_set[i],schma +
        '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION') | ((schma +
        '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(pdr)) );
  END_REPEAT;
  pdrel_set := bag_to_set(USEDIN(assy.related_product_definition,schma +
       '.PRODUCT_DEFINITION_RELATIONSHIP.' +
       'RELATED_PRODUCT_DEFINITION'));
  REPEAT j := 1 TO HIINDEX(pdrel_set) BY 1;
    pr2_set := pr2_set + USEDIN(pdrel_set[j],schma +
        '.PROPERTY_DEFINITION.DEFINITION');
  END_REPEAT;
  REPEAT i := 1 TO HIINDEX(pr2_set) BY 1;
    sdr_set := sdr_set + QUERY ( pdr <* USEDIN(pr2_set[i],schma +
        '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION') | ((schma +
        '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(pdr)) );
```

```
    END_REPEAT;
    IF SIZEOF(sdr_set) > 0 THEN
      REPEAT i := 1 TO HIINDEX(sdr_set) BY 1;
        srr_set := QUERY ( rr <* bag_to_set(USEDIN(sdr_set[i]\
            property_definition_representation.used_representation,schma +
            '.REPRESENTATION_RELATIONSHIP.REP_2')) | ((schma +
            '.SHAPE_REPRESENTATION_RELATIONSHIP') IN TYPEOF(rr)) );
        IF SIZEOF(srr_set) > 0 THEN
          REPEAT j := 1 TO HIINDEX(srr_set) BY 1;
            IF SIZEOF(QUERY ( pdr <* bag_to_set(USEDIN(srr_set[j]\
                representation_relationship.rep_1,schma +
                '.PROPERTY_DEFINITION_REPRESENTATION.USED_REPRESENTATION'))
                 | ((schma + '.SHAPE_DEFINITION_REPRESENTATION') IN TYPEOF(
                pdr)) ) * QUERY ( pdr <* bag_to_set(USEDIN(assy.
                relating_product_definition,schma +
                '.PROPERTY_DEFINITION_REPRESENTATION.DEFINITION')) | ((
                schma + '.SHAPE_DEFINITION_REPRESENTATION') IN
                TYPEOF(pdr)) )) >= 1 THEN
              IF SIZEOF(QUERY ( cdsr <* USEDIN(srr_set[j],schma +
                  '.CONTEXT_DEPENDENT_SHAPE_REPRESENTATION.' +
                  'REPRESENTATION_RELATION') | (NOT (cdsr\
                  context_dependent_shape_representation.
                  represented_product_relation\property_definition.
                  definition :=: assy)) )) > 0 THEN
                RETURN(FALSE);
              END_IF;
            END_IF;
          END_REPEAT;
        END_IF;
      END_REPEAT;
    END_IF;
    RETURN(TRUE);
  END_FUNCTION; -- assembly_shape_is_defined

  FUNCTION associated_surface(arg: pcurve_or_surface): surface;
    LOCAL
      surf : surface;
    END_LOCAL;
    IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(arg) THEN
      surf := arg.basis_surface;
    ELSE
      surf := arg;
```

```
    END_IF;
    RETURN(surf);
  END_FUNCTION; -- associated_surface

  FUNCTION bag_to_set(the_bag: BAG OF GENERIC:intype
      ): SET OF GENERIC:intype;
    LOCAL
      i       : INTEGER;
      the_set : SET OF GENERIC:intype := [];
    END_LOCAL;
    IF SIZEOF(the_bag) > 0 THEN
      REPEAT i := 1 TO HIINDEX(the_bag) BY 1;
        the_set := the_set + the_bag[i];
      END_REPEAT;
    END_IF;
    RETURN(the_set);
  END_FUNCTION; -- bag_to_set

  FUNCTION base_axis(dim: INTEGER; axis1, axis2, axis3: direction
      ): LIST [2:3] OF direction;
    LOCAL
      u      : LIST [2:3] OF direction;
      d1     : direction;
      d2     : direction;
      factor : REAL;
    END_LOCAL;
    IF dim = 3 THEN
      d1 := NVL(normalise(axis3),dummy_gri || direction([0,0,1]));
      d2 := first_proj_axis(d1,axis1);
      u := [d2,second_proj_axis(d1,d2,axis2),d1];
    ELSE
      IF EXISTS(axis1) THEN
        d1 := normalise(axis1);
        u := [d1,orthogonal_complement(d1)];
        IF EXISTS(axis2) THEN
          factor := dot_product(axis2,u[2]);
          IF factor < 0 THEN
            u[2].direction_ratios[1] := -u[2].direction_ratios[1];
            u[2].direction_ratios[2] := -u[2].direction_ratios[2];
          END_IF;
        END_IF;
      ELSE
```

```
      IF EXISTS(axis2) THEN
        d1 := normalise(axis2);
        u := [orthogonal_complement(d1),d1];
        u[1].direction_ratios[1] := -u[1].direction_ratios[1];
        u[1].direction_ratios[2] := -u[1].direction_ratios[2];
      ELSE
        u := [dummy_gri || direction([1,0]),dummy_gri ||
        direction([0,1])];
      END_IF;
    END_IF;
  END_IF;
  RETURN(u);
END_FUNCTION; -- base_axis

FUNCTION boolean_choose(b: BOOLEAN;
             choice1, choice2: GENERIC:item): GENERIC:item;
  IF b THEN
    RETURN(choice1);
  ELSE
    RETURN(choice2);
  END_IF;
END_FUNCTION; -- boolean_choose

FUNCTION build_2axes(ref_direction: direction): LIST [2:2] OF direction;
  LOCAL
    d : direction := NVL(normalise(ref_direction),dummy_gri ||
         direction([1,0]));
  END_LOCAL;
  RETURN([d,orthogonal_complement(d)]);
END_FUNCTION; -- build_2axes

FUNCTION build_axes(axis, ref_direction: direction
    ): LIST [3:3] OF direction;
  LOCAL
    d1 : direction;
    d2 : direction;
  END_LOCAL;
  d1 := NVL(normalise(axis),dummy_gri || direction([0,0,1]));
  d2 := first_proj_axis(d1,ref_direction);
  RETURN([d2,normalise(cross_product(d1,d2)).orientation,d1]);
END_FUNCTION; -- build_axes
```

```
FUNCTION cc_design_date_time_correlation
  (e : cc_design_date_and_time_assignment ) : BOOLEAN;
  LOCAL
    dt_role : STRING;
  END_LOCAL;
    dt_role := e\date_and_time_assignment.role.name;
  CASE dt_role OF
    'creation_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
    'request_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE_REQUEST',
                                'CONFIG_CONTROL_DESIGN.START_REQUEST'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'release_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE',
                                'CONFIG_CONTROL_DESIGN.START_WORK'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'start_date'          : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (
                                ['CONFIG_CONTROL_DESIGN.CHANGE',
                                'CONFIG_CONTROL_DESIGN.START_WORK'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
    'sign_off_date'         : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'APPROVAL_PERSON_ORGANIZATION'
```

```
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
     'contract_date'         : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.CONTRACT'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
     'certification_date'    : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.CERTIFICATION'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
     'classification_date'   : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'SECURITY_CLASSIFICATION'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
     'declassification_date' : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'SECURITY_CLASSIFICATION'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
     OTHERWISE : RETURN(TRUE);
   END_CASE;
   RETURN (TRUE);
 END_FUNCTION;  -- cc_design_date_time_correlation

 FUNCTION cc_design_person_and_organization_correlation
    (e : cc_design_person_and_organization_assignment ) : BOOLEAN;
    LOCAL
      po_role : STRING;
    END_LOCAL;
      po_role := e\person_and_organization_assignment.role.name;
    CASE po_role OF
     'request_recipient'      : IF SIZEOF (e.items) <>
```

```
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF(['CONFIG_CONTROL_DESIGN.' +
                                'CHANGE_REQUEST',
                                'CONFIG_CONTROL_DESIGN.' +
                                'START_REQUEST'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
'initiator'                 : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF(['CONFIG_CONTROL_DESIGN.' +
                                'CHANGE_REQUEST',
                                'CONFIG_CONTROL_DESIGN.' +
                                'START_REQUEST',
                                'CONFIG_CONTROL_DESIGN.' +
                                'START_WORK',
                                'CONFIG_CONTROL_DESIGN.' +
                                'CHANGE'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN(FALSE);
                              END_IF;
'creator'                   : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                SIZEOF (['CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION_FORMATION',
                                'CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION'] *
                                TYPEOF (x)) = 1))
                                THEN RETURN (FALSE);
                              END_IF;
'part_supplier'             : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION_FORMATION'
                                IN TYPEOF (x)))
                                THEN RETURN(FALSE);
                              END_IF;
'design_supplier'           : IF SIZEOF (e.items) <>
                                SIZEOF (QUERY (x <* e.items |
                                'CONFIG_CONTROL_DESIGN.' +
                                'PRODUCT_DEFINITION_FORMATION'
                                IN TYPEOF (x)))
```

```
                                  THEN RETURN(FALSE);
                                END_IF;
      'design_owner'           : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.PRODUCT'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
      'configuration_manager'  : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'CONFIGURATION_ITEM'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
      'contractor'             : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.CONTRACT'
                                  IN TYPEOF (x)))
                                  THEN RETURN(FALSE);
                                END_IF;
      'classification_officer' : IF SIZEOF (e.items) <>
                                  SIZEOF (QUERY (x <* e.items |
                                  'CONFIG_CONTROL_DESIGN.' +
                                  'SECURITY_CLASSIFICATION'
                                  IN TYPEOF (x))) THEN
                                  RETURN(FALSE);
                                END_IF;
      OTHERWISE : RETURN(TRUE);
    END_CASE;
    RETURN (TRUE);
  END_FUNCTION; -- cc_design_person_and_organization_correlation

  FUNCTION closed_shell_reversed(a_shell: closed_shell
      ): oriented_closed_shell;
    LOCAL
      the_reverse : oriented_closed_shell;
    END_LOCAL;
    IF 'CONFIG_CONTROL_DESIGN.ORIENTED_CLOSED_SHELL' IN TYPEOF(a_shell)
         THEN
      the_reverse := dummy_tri || connected_face_set(a_shell\
          connected_face_set.cfs_faces) || closed_shell() ||
```

```
        oriented_closed_shell(a_shell\oriented_closed_shell.
        closed_shell_element,NOT a_shell\oriented_closed_shell.
        orientation);
  ELSE
    the_reverse := dummy_tri || connected_face_set(a_shell\
        connected_face_set.cfs_faces) || closed_shell() ||
        oriented_closed_shell(a_shell,FALSE);
  END_IF;
  RETURN(the_reverse);
END_FUNCTION; -- closed_shell_reversed

FUNCTION conditional_reverse(p: BOOLEAN;
             an_item: reversible_topology): reversible_topology;
  IF p THEN RETURN(an_item);
  ELSE
    RETURN(topology_reversed(an_item));
  END_IF;
END_FUNCTION; -- conditional_reverse

FUNCTION constraints_composite_curve_on_surface(
             c: composite_curve_on_surface): BOOLEAN;
  LOCAL
    n_segments : INTEGER := SIZEOF(c.segments);
  END_LOCAL;
  REPEAT k := 1 TO n_segments BY 1;
    IF (NOT ('CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(c\composite_curve.
        segments[k].parent_curve))) AND (NOT (
        'CONFIG_CONTROL_DESIGN.SURFACE_CURVE' IN TYPEOF(c\composite_curve
        .segments[k].parent_curve))) AND (NOT (
        'CONFIG_CONTROL_DESIGN.COMPOSITE_CURVE_ON_SURFACE' IN TYPEOF(c\
        composite_curve.segments[k].parent_curve))) THEN
      RETURN(FALSE);
    END_IF;
  END_REPEAT;
  RETURN(TRUE);
END_FUNCTION; -- constraints_composite_curve_on_surface

FUNCTION constraints_geometry_shell_based_surface_model(
             m: shell_based_surface_model): BOOLEAN;
  LOCAL
    result : BOOLEAN := TRUE;
  END_LOCAL;
```

```
    REPEAT j := 1 TO SIZEOF(m.sbsm_boundary) BY 1;
      IF (NOT ('CONFIG_CONTROL_DESIGN.OPEN_SHELL' IN TYPEOF(m.
          sbsm_boundary[j]))) AND (NOT (
          'CONFIG_CONTROL_DESIGN.CLOSED_SHELL' IN
           TYPEOF(m.sbsm_boundary[j]))) THEN
        result := FALSE;
        RETURN(result);
      END_IF;
    END_REPEAT;
    RETURN(result);
  END_FUNCTION; -- constraints_geometry_shell_based_surface_model

  FUNCTION constraints_geometry_shell_based_wireframe_model(
               m: shell_based_wireframe_model): BOOLEAN;
    LOCAL
      result : BOOLEAN := TRUE;
    END_LOCAL;
    REPEAT j := 1 TO SIZEOF(m.sbwm_boundary) BY 1;
      IF (NOT ('CONFIG_CONTROL_DESIGN.WIRE_SHELL' IN TYPEOF(m.
          sbwm_boundary[j]))) AND (NOT (
          'CONFIG_CONTROL_DESIGN.VERTEX_SHELL' IN
          TYPEOF(m.sbwm_boundary[j])))
          THEN result := FALSE;
        RETURN(result);
      END_IF;
    END_REPEAT;
    RETURN(result);
  END_FUNCTION; -- constraints_geometry_shell_based_wireframe_model

  FUNCTION constraints_param_b_spline(degree, up_knots, up_cp: INTEGER;
               knot_mult: LIST OF INTEGER;
               knots: LIST OF parameter_value): BOOLEAN;
    LOCAL
      k      : INTEGER;
      sum    : INTEGER;
      result : BOOLEAN := TRUE;
    END_LOCAL;
    sum := knot_mult[1];
    REPEAT i := 2 TO up_knots BY 1;
      sum := sum + knot_mult[i];
    END_REPEAT;
    IF (degree < 1) OR (up_knots < 2) OR (up_cp < degree) OR (sum <> (
```

```
      degree + up_cp + 2)) THEN result := FALSE;
    RETURN(result);
  END_IF;
  k := knot_mult[1];
  IF (k < 1) OR (k > (degree + 1)) THEN result := FALSE;
    RETURN(result);
  END_IF;
  REPEAT i := 2 TO up_knots BY 1;
    IF (knot_mult[i] < 1) OR (knots[i] <= knots[i - 1]) THEN
      result := FALSE;
      RETURN(result);
    END_IF;
    k := knot_mult[i];
    IF (i < up_knots) AND (k > degree) THEN
      result := FALSE;
      RETURN(result);
    END_IF;
    IF (i = up_knots) AND (k > (degree + 1)) THEN
      result := FALSE;
      RETURN(result);
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION; -- constraints_param_b_spline

FUNCTION constraints_rectangular_composite_surface(
             s: rectangular_composite_surface): BOOLEAN;
  REPEAT i := 1 TO s.n_u BY 1;
    REPEAT j := 1 TO s.n_v BY 1;
      IF NOT (('CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE' IN TYPEOF(s.
          segments[i][j].parent_surface)) OR (
          'CONFIG_CONTROL_DESIGN.RECTANGULAR_TRIMMED_SURFACE' IN TYPEOF(s
          .segments[i][j].parent_surface))) THEN
        RETURN(FALSE);
      END_IF;
    END_REPEAT;
  END_REPEAT;
  REPEAT i := 1 TO s.n_u - 1 BY 1;
    REPEAT j := 1 TO s.n_v BY 1;
      IF s.segments[i][j].u_transition = discontinuous THEN
        RETURN(FALSE);
      END_IF;
```

```
      END_REPEAT;
    END_REPEAT;
    REPEAT i := 1 TO s.n_u BY 1;
      REPEAT j := 1 TO s.n_v - 1 BY 1;
        IF s.segments[i][j].v_transition = discontinuous THEN
          RETURN(FALSE);
        END_IF;
      END_REPEAT;
    END_REPEAT;
    RETURN(TRUE);
  END_FUNCTION; -- constraints_rectangular_composite_surface

  FUNCTION cross_product(arg1, arg2: direction): vector;
    LOCAL
      v2     : LIST [3:3] OF REAL;
      v1     : LIST [3:3] OF REAL;
      mag    : REAL;
      res    : direction;
      result : vector;
    END_LOCAL;
    IF (NOT EXISTS(arg1)) OR (arg1.dim = 2) OR (NOT EXISTS(arg2)) OR (arg2
        .dim = 2) THEN RETURN(?);
    ELSE
      BEGIN
        v1 := normalise(arg1).direction_ratios;
        v2 := normalise(arg2).direction_ratios;
        res := dummy_gri || direction([(v1[2] * v2[3]) - (v1[3] * v2[2]),(
            v1[3] * v2[1]) - (v1[1] * v2[3]),(v1[1] * v2[2]) - (v1[2] * v2[
            1])]);
        mag := 0;
        REPEAT i := 1 TO 3 BY 1;
          mag := mag + (res.direction_ratios[i] * res.direction_ratios[i]);
        END_REPEAT;
        IF mag > 0 THEN
          result := dummy_gri || vector(res,SQRT(mag));
        ELSE
          result := dummy_gri || vector(arg1,0);
        END_IF;
        RETURN(result);
      END;
    END_IF;
  END_FUNCTION; -- cross_product
```

```
FUNCTION curve_weights_positive(b: rational_b_spline_curve): BOOLEAN;
  LOCAL
    result : BOOLEAN := TRUE;
  END_LOCAL;
  REPEAT i := 0 TO b.upper_index_on_control_points BY 1;
    IF b.weights[i] <= 0 THEN result := FALSE;
      RETURN(result);
    END_IF;
  END_REPEAT;
  RETURN(result);
END_FUNCTION; -- curve_weights_positive

FUNCTION derive_dimensional_exponents(x: unit): dimensional_exponents;
  LOCAL
    i      : INTEGER;
    result : dimensional_exponents := dimensional_exponents(0,0,0,0,0,0,
               0);
  END_LOCAL;
  IF 'CONFIG_CONTROL_DESIGN.DERIVED_UNIT' IN TYPEOF(x) THEN
    REPEAT i := LOINDEX(x.elements) TO HIINDEX(x.elements) BY 1;
          result.length_exponent := result.length_exponent +
          (x.elements[i].
          exponent * x.elements[i].unit.dimensions.length_exponent);
          result.mass_exponent := result.mass_exponent + (x.elements[i].
          exponent * x.elements[i].unit.dimensions.mass_exponent);
          result.time_exponent := result.time_exponent + (x.elements[i].
          exponent * x.elements[i].unit.dimensions.time_exponent);
          result.electric_current_exponent := result.
          electric_current_exponent + (x.elements[i].exponent * x.
          elements[i].unit.dimensions.electric_current_exponent);
          result.thermodynamic_temperature_exponent := result.
          thermodynamic_temperature_exponent + (x.elements[i].exponent *
          x.elements[i].unit.dimensions.
          thermodynamic_temperature_exponent);
          result.amount_of_substance_exponent := result.
          amount_of_substance_exponent + (x.elements[i].exponent * x.
          elements[i].unit.dimensions.amount_of_substance_exponent);
          result.luminous_intensity_exponent := result.
          luminous_intensity_exponent + (x.elements[i].exponent * x.
          elements[i].unit.dimensions.luminous_intensity_exponent);
    END_REPEAT;
  ELSE
```

```
    result := x.dimensions;
  END_IF;
  RETURN(result);
END_FUNCTION; -- derive_dimensional_exponents

FUNCTION dimension_of(item: geometric_representation_item
    ): dimension_count;
  LOCAL
    x : SET OF representation;
    y : representation_context;
  END_LOCAL;
  x := using_representations(item);
  y := x[1].context_of_items;
  RETURN(y\geometric_representation_context.coordinate_space_dimension);
END_FUNCTION; -- dimension_of

FUNCTION dimensions_for_si_unit(n: si_unit_name
    ): dimensional_exponents;
  CASE n OF
    metre          :    RETURN(dimensional_exponents(1,0,0,0,0,0,0));
    gram           :    RETURN(dimensional_exponents(0,1,0,0,0,0,0));
    second         :    RETURN(dimensional_exponents(0,0,1,0,0,0,0));
    ampere         :    RETURN(dimensional_exponents(0,0,0,1,0,0,0));
    kelvin         :    RETURN(dimensional_exponents(0,0,0,0,1,0,0));
    mole           :    RETURN(dimensional_exponents(0,0,0,0,0,1,0));
    candela        :    RETURN(dimensional_exponents(0,0,0,0,0,0,1));
    radian         :    RETURN(dimensional_exponents(0,0,0,0,0,0,0));
    steradian      :    RETURN(dimensional_exponents(0,0,0,0,0,0,0));
    hertz          :    RETURN(dimensional_exponents(0,0,-1,0,0,0,0));
    newton         :    RETURN(dimensional_exponents(1,1,-2,0,0,0,0));
    pascal         :    RETURN(dimensional_exponents(-1,1,-2,0,0,0,0));
    joule          :    RETURN(dimensional_exponents(2,1,-2,0,0,0,0));
    watt           :    RETURN(dimensional_exponents(2,1,-3,0,0,0,0));
    coulomb        :    RETURN(dimensional_exponents(0,0,1,1,0,0,0));
    volt           :    RETURN(dimensional_exponents(2,1,-3,-1,0,0,0));
    farad          :    RETURN(dimensional_exponents(-2,-1,4,1,0,0,0));
    ohm            :    RETURN(dimensional_exponents(2,1,-3,-2,0,0,0));
    siemens        :    RETURN(dimensional_exponents(-2,-1,3,2,0,0,0));
    weber          :    RETURN(dimensional_exponents(2,1,-2,-1,0,0,0));
    tesla          :    RETURN(dimensional_exponents(0,1,-2,-1,0,0,0));
    henry          :    RETURN(dimensional_exponents(2,1,-2,-2,0,0,0));
    degree_celsius :    RETURN(dimensional_exponents(0,0,0,0,1,0,0));
```

```
    lumen          :    RETURN(dimensional_exponents(0,0,0,0,0,0,1));
    lux            :    RETURN(dimensional_exponents(-2,0,0,0,0,0,1));
    becquerel      :    RETURN(dimensional_exponents(0,0,-1,0,0,0,0));
    gray           :    RETURN(dimensional_exponents(2,0,-2,0,0,0,0));
    sievert        :    RETURN(dimensional_exponents(2,0,-2,0,0,0,0));
    END_CASE;
END_FUNCTION; -- dimensions_for_si_unit

FUNCTION dot_product(arg1, arg2: direction): REAL;
  LOCAL
    ndim   : INTEGER;
    scalar : REAL;
    vec1   : direction;
    vec2   : direction;
  END_LOCAL;
  IF (NOT EXISTS(arg1)) OR (NOT EXISTS(arg2)) THEN
    scalar := ?;
  ELSE
    IF arg1.dim <> arg2.dim THEN scalar := ?;
    ELSE
      BEGIN
        vec1 := normalise(arg1);
        vec2 := normalise(arg2);
        ndim := arg1.dim;
        scalar := 0;
        REPEAT i := 1 TO ndim BY 1;
          scalar := scalar + (vec1.direction_ratios[i] * vec2.
              direction_ratios[i]);
        END_REPEAT;
      END;
    END_IF;
  END_IF;
  RETURN(scalar);
END_FUNCTION; -- dot_product

FUNCTION edge_reversed(an_edge: edge): oriented_edge;
  LOCAL
    the_reverse : oriented_edge;
  END_LOCAL;
  IF 'CONFIG_CONTROL_DESIGN.ORIENTED_EDGE' IN TYPEOF(an_edge) THEN
    the_reverse := dummy_tri || edge(an_edge.edge_end,an_edge.edge_start)
        || oriented_edge(an_edge\oriented_edge.edge_element,NOT an_edge\
```

```
        oriented_edge.orientation);
  ELSE
    the_reverse := dummy_tri || edge(an_edge.edge_end,an_edge.edge_start)
        || oriented_edge(an_edge,FALSE);
  END_IF;
  RETURN(the_reverse);
END_FUNCTION; -- edge_reversed

FUNCTION face_bound_reversed(a_face_bound: face_bound): face_bound;
  LOCAL
    the_reverse : face_bound;
  END_LOCAL;
  IF 'CONFIG_CONTROL_DESIGN.FACE_OUTER_BOUND' IN TYPEOF(a_face_bound)
       THEN
    the_reverse := dummy_tri || face_bound(a_face_bound\face_bound.bound,
        NOT a_face_bound\face_bound.orientation) || face_outer_bound();
  ELSE
    the_reverse := dummy_tri || face_bound(a_face_bound.bound,NOT
        a_face_bound.orientation);
  END_IF;
  RETURN(the_reverse);
END_FUNCTION; -- face_bound_reversed

FUNCTION face_reversed(a_face: face): oriented_face;
  LOCAL
    the_reverse : oriented_face;
  END_LOCAL;
  IF 'CONFIG_CONTROL_DESIGN.ORIENTED_FACE' IN TYPEOF(a_face) THEN
    the_reverse := dummy_tri || face(set_of_topology_reversed(a_face.
        bounds)) || oriented_face(a_face\oriented_face.face_element,NOT
        a_face\oriented_face.orientation);
  ELSE
    the_reverse := dummy_tri || face(set_of_topology_reversed(a_face.
        bounds)) || oriented_face(a_face,FALSE);
  END_IF;
  RETURN(the_reverse);
END_FUNCTION; -- face_reversed

FUNCTION first_proj_axis(z_axis, arg: direction): direction;
  LOCAL
    x_vec  : vector;
    v      : direction;
```

```
    z      : direction;
    x_axis : direction;
  END_LOCAL;
  IF NOT EXISTS(z_axis) THEN RETURN(?);
  ELSE
    z := normalise(z_axis);
    IF NOT EXISTS(arg) THEN
      IF z.direction_ratios <> [1,0,0] THEN
        v := dummy_gri || direction([1,0,0]);
      ELSE
        v := dummy_gri || direction([0,1,0]);
      END_IF;
    ELSE
      IF arg.dim <> 3 THEN RETURN(?);
      END_IF;
      IF cross_product(arg,z).magnitude = 0 THEN RETURN(?);
      ELSE
        v := normalise(arg);
      END_IF;
    END_IF;
    x_vec := scalar_times_vector(dot_product(v,z),z);
    x_axis := vector_difference(v,x_vec).orientation;
    x_axis := normalise(x_axis);
  END_IF;
  RETURN(x_axis);
END_FUNCTION; -- first_proj_axis

FUNCTION gbsf_check_curve(cv: curve): BOOLEAN;
  IF SIZEOF(['CONFIG_CONTROL_DESIGN.BOUNDED_CURVE',
      'CONFIG_CONTROL_DESIGN.CONIC',
      'CONFIG_CONTROL_DESIGN.CURVE_REPLICA',
      'CONFIG_CONTROL_DESIGN.LINE',
      'CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D'] * TYPEOF(cv)) > 1 THEN
    RETURN(FALSE);
  ELSE
    IF SIZEOF(['CONFIG_CONTROL_DESIGN.CIRCLE',
        'CONFIG_CONTROL_DESIGN.ELLIPSE'] * TYPEOF(cv)) = 1 THEN
      RETURN(TRUE);
    ELSE
      IF (('CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE' IN TYPEOF(cv)) AND (cv
          \b_spline_curve.self_intersect = FALSE)) OR (cv\b_spline_curve.
          self_intersect = UNKNOWN) THEN
```

```
      RETURN(TRUE);
    ELSE
      IF (('CONFIG_CONTROL_DESIGN.COMPOSITE_CURVE' IN TYPEOF(cv)) AND
          (cv\composite_curve.self_intersect = FALSE)) OR (cv\
          composite_curve.self_intersect = UNKNOWN) THEN
        RETURN(SIZEOF(QUERY ( seg <* cv\composite_curve.segments | (
            NOT gbsf_check_curve(seg.parent_curve)) )) = 0);
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.CURVE_REPLICA' IN TYPEOF(cv) THEN
          RETURN(gbsf_check_curve(cv\curve_replica.parent_curve));
        ELSE
          IF ('CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D' IN TYPEOF(cv))
              AND ((cv\offset_curve_3d.self_intersect = FALSE) OR (cv\
              offset_curve_3d.self_intersect = UNKNOWN)) AND (NOT (
              'CONFIG_CONTROL_DESIGN.POLYLINE' IN
              TYPEOF(cv.basis_curve))) THEN
            RETURN(gbsf_check_curve(cv\offset_curve_3d.basis_curve));
          ELSE
            IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(cv) THEN
              RETURN(gbsf_check_curve(cv\pcurve.reference_to_curve\
                  representation.items[1]) AND gbsf_check_surface(cv\
                  pcurve.basis_surface));
            ELSE
              IF 'CONFIG_CONTROL_DESIGN.POLYLINE' IN TYPEOF(cv) THEN
                IF SIZEOF(cv\polyline.points) >= 3 THEN
                  RETURN(TRUE);
                END_IF;
              ELSE
                IF 'CONFIG_CONTROL_DESIGN.SURFACE_CURVE' IN TYPEOF(cv)
                     THEN
                  IF gbsf_check_curve(cv\surface_curve.curve_3d) THEN
                    REPEAT i := 1 TO SIZEOF(cv\surface_curve.
                        associated_geometry) BY 1;
                      IF 'CONFIG_CONTROL_DESIGN.SURFACE' IN TYPEOF(cv\
                          surface_curve.associated_geometry[i]) THEN
                        IF NOT gbsf_check_surface(cv\surface_curve.
                            associated_geometry[i]) THEN
                          RETURN(FALSE);
                        END_IF;
                      ELSE
                        IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(cv
                            \surface_curve.associated_geometry[i])
```

```
                                THEN
                              IF NOT gbsf_check_curve(cv\surface_curve.
                                  associated_geometry[i]) THEN
                                RETURN(FALSE);
                              END_IF;
                            END_IF;
                          END_IF;
                        END_REPEAT;
                        RETURN(TRUE);
                      END_IF;
                    ELSE
                      IF 'CONFIG_CONTROL_DESIGN.TRIMMED_CURVE' IN TYPEOF(
                          cv) THEN
                        IF SIZEOF(['CONFIG_CONTROL_DESIGN.LINE',
                            'CONFIG_CONTROL_DESIGN.PARABOLA',
                            'CONFIG_CONTROL_DESIGN.HYPERBOLA'] * TYPEOF(cv\
                            trimmed_curve.basis_curve)) = 1 THEN
                          RETURN(TRUE);
                        ELSE
                          RETURN(gbsf_check_curve(cv\trimmed_curve.
                              basis_curve));
                        END_IF;
                      END_IF;
                    END_IF;
                  END_IF;
                END_IF;
              END_IF;
            END_IF;
          END_IF;
        END_IF;
      END_IF;
    END_IF;
    RETURN(FALSE);
  END_FUNCTION; -- gbsf_check_curve

  FUNCTION gbsf_check_point(pnt: point): BOOLEAN;
    IF 'CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN TYPEOF(pnt) THEN
      RETURN(TRUE);
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.POINT_ON_CURVE' IN TYPEOF(pnt) THEN
        RETURN(gbsf_check_curve(pnt\point_on_curve.basis_curve));
      ELSE
```

```
      IF 'CONFIG_CONTROL_DESIGN.POINT_ON_SURFACE' IN TYPEOF(pnt) THEN
        RETURN(gbsf_check_surface(pnt\point_on_surface.basis_surface));
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.DEGENERATE_PCURVE' IN TYPEOF(pnt)
             THEN
          RETURN(gbsf_check_curve(pnt\degenerate_pcurve.
              reference_to_curve\representation.items[1]) AND
              gbsf_check_surface(pnt\degenerate_pcurve.basis_surface));
        END_IF;
      END_IF;
    END_IF;
  END_IF;
  RETURN(FALSE);
END_FUNCTION; -- gbsf_check_point

FUNCTION gbsf_check_surface(sf: surface): BOOLEAN;
  IF (('CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE' IN TYPEOF(sf)) AND (sf\
      b_spline_surface.self_intersect = FALSE)) OR (sf\b_spline_surface.
      self_intersect = UNKNOWN) THEN RETURN(TRUE);
  ELSE
    IF SIZEOF(['CONFIG_CONTROL_DESIGN.SPHERICAL_SURFACE',
        'CONFIG_CONTROL_DESIGN.TOROIDAL_SURFACE'] * TYPEOF(sf)) = 1 THEN
      RETURN(TRUE);
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.CURVE_BOUNDED_SURFACE' IN TYPEOF(sf)
           THEN
        IF SIZEOF(['CONFIG_CONTROL_DESIGN.CONICAL_SURFACE',
            'CONFIG_CONTROL_DESIGN.CYLINDRICAL_SURFACE',
            'CONFIG_CONTROL_DESIGN.PLANE'] * TYPEOF(sf\
            curve_bounded_surface.basis_surface)) = 1 THEN
          RETURN(SIZEOF(QUERY ( bcurve <* sf\curve_bounded_surface.
              boundaries | (NOT gbsf_check_curve(bcurve)) )) = 0);
        ELSE
          IF gbsf_check_surface(sf\curve_bounded_surface.basis_surface)
               THEN
            RETURN(SIZEOF(QUERY ( bcurve <* sf\curve_bounded_surface.
                boundaries | (NOT gbsf_check_curve(bcurve)) )) = 0);
          END_IF;
        END_IF;
      ELSE
        IF (('CONFIG_CONTROL_DESIGN.OFFSET_SURFACE' IN TYPEOF(sf)) AND (
            sf\offset_surface.self_intersect = FALSE)) OR (sf\
```

```
    offset_surface.self_intersect = UNKNOWN) THEN
  RETURN(gbsf_check_surface(sf\offset_surface.basis_surface));
ELSE
  IF 'CONFIG_CONTROL_DESIGN.RECTANGULAR_COMPOSITE_SURFACE' IN
      TYPEOF(sf) THEN
    REPEAT i := 1 TO SIZEOF(sf\rectangular_composite_surface.
        segments) BY 1;
      REPEAT j := 1 TO SIZEOF(sf\rectangular_composite_surface.
          segments[i]) BY 1;
        IF NOT gbsf_check_surface(sf\
            rectangular_composite_surface.segments[i][j].
            parent_surface) THEN RETURN(FALSE);
        END_IF;
      END_REPEAT;
    END_REPEAT;
    RETURN(TRUE);
  ELSE
    IF 'CONFIG_CONTROL_DESIGN.RECTANGULAR_TRIMMED_SURFACE' IN
        TYPEOF(sf) THEN
      IF SIZEOF(['CONFIG_CONTROL_DESIGN.CONICAL_SURFACE',
          'CONFIG_CONTROL_DESIGN.CYLINDRICAL_SURFACE',
          'CONFIG_CONTROL_DESIGN.PLANE'] * TYPEOF(sf\
          rectangular_trimmed_surface.basis_surface)) = 1 THEN
        RETURN(TRUE);
      ELSE
        RETURN(gbsf_check_surface(sf\rectangular_trimmed_surface
            .basis_surface));
      END_IF;
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.SURFACE_REPLICA' IN TYPEOF(sf)
           THEN
        RETURN(gbsf_check_surface(sf\surface_replica.
            parent_surface));
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.SWEPT_SURFACE' IN TYPEOF(sf)
             THEN
          RETURN(gbsf_check_curve(sf\swept_surface.swept_curve));
        END_IF;
      END_IF;
    END_IF;
  END_IF;
END_IF;
```

```
      END_IF;
    END_IF;
  END_IF;
  RETURN(FALSE);
END_FUNCTION; -- gbsf_check_surface

FUNCTION get_basis_surface(c: curve_on_surface): SET [0:2] OF surface;
  LOCAL
    surfs : SET [0:2] OF surface;
    n     : INTEGER;
  END_LOCAL;
  surfs := [];
  IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(c) THEN
    surfs := [c\pcurve.basis_surface];
  ELSE
    IF 'CONFIG_CONTROL_DESIGN.SURFACE_CURVE' IN TYPEOF(c) THEN
      n := SIZEOF(c\surface_curve.associated_geometry);
      REPEAT i := 1 TO n BY 1;
        surfs := surfs + associated_surface(c\surface_curve.
            associated_geometry[i]);
      END_REPEAT;
    END_IF;
  END_IF;
  IF 'CONFIG_CONTROL_DESIGN.COMPOSITE_CURVE_ON_SURFACE' IN TYPEOF(c)
       THEN
    n := SIZEOF(c\composite_curve.segments);
    surfs := get_basis_surface(c\composite_curve.segments[1].
        parent_curve);
    IF n > 1 THEN
      REPEAT i := 2 TO n BY 1;
        surfs := surfs * get_basis_surface(c\composite_curve.segments[i]
            .parent_curve);
      END_REPEAT;
    END_IF;
  END_IF;
  RETURN(surfs);
END_FUNCTION; -- get_basis_surface

FUNCTION item_in_context(item: representation_item;
             cntxt: representation_context): BOOLEAN;
  LOCAL
    i : INTEGER;
```

```
    y : BAG OF representation_item;
  END_LOCAL;
  IF SIZEOF(USEDIN(item,'CONFIG_CONTROL_DESIGN.REPRESENTATION.ITEMS') *
      cntxt.representations_in_context) > 0 THEN RETURN(TRUE);
  ELSE
    y := QUERY ( z <* USEDIN(item,'') | (
        'CONFIG_CONTROL_DESIGN.REPRESENTATION_ITEM' IN TYPEOF(z)) );
    IF SIZEOF(y) > 0 THEN
      REPEAT i := 1 TO HIINDEX(y) BY 1;
        IF item_in_context(y[i],cntxt) THEN RETURN(TRUE);
        END_IF;
      END_REPEAT;
    END_IF;
  END_IF;
  RETURN(FALSE);
END_FUNCTION; -- item_in_context

FUNCTION leap_year(year: year_number): BOOLEAN;
  IF (((year MOD 4) = 0) AND ((year MOD 100) <> 0)) OR ((year MOD 400) =
       0) THEN RETURN(TRUE);
  ELSE
    RETURN(FALSE);
  END_IF;
END_FUNCTION; -- leap_year

FUNCTION list_face_loops(f: face): LIST [0:?] OF loop;
  LOCAL
    loops : LIST [0:?] OF loop := [];
  END_LOCAL;
  REPEAT i := 1 TO SIZEOF(f.bounds) BY 1;
    loops := loops + f.bounds[i].bound;
  END_REPEAT;
  RETURN(loops);
END_FUNCTION; -- list_face_loops

FUNCTION list_of_topology_reversed(
             a_list: list_of_reversible_topology_item
    ): list_of_reversible_topology_item;
  LOCAL
    the_reverse : list_of_reversible_topology_item;
  END_LOCAL;
  the_reverse := [];
```

```
    REPEAT i := 1 TO SIZEOF(a_list) BY 1;
      the_reverse := topology_reversed(a_list[i]) + the_reverse;
    END_REPEAT;
    RETURN(the_reverse);
  END_FUNCTION; -- list_of_topology_reversed

  FUNCTION list_to_array(lis: LIST [0:?] OF GENERIC:t;
                 low, u: INTEGER): ARRAY OF GENERIC:t;
    LOCAL
      n   : INTEGER;
      res : ARRAY [low:u] OF GENERIC:t;
    END_LOCAL;
    n := SIZEOF(lis);
    IF n <> ((u - low) + 1) THEN RETURN(?);
    ELSE
      res := [lis[1],n];
      REPEAT i := 2 TO n BY 1;
        res[(low + i) - 1] := lis[i];
      END_REPEAT;
      RETURN(res);
    END_IF;
  END_FUNCTION; -- list_to_array

  FUNCTION list_to_set(l: LIST [0:?] OF GENERIC:t): SET OF GENERIC:t;
    LOCAL
      s : SET OF GENERIC:t := [];
    END_LOCAL;
    REPEAT i := 1 TO SIZEOF(l) BY 1;
      s := s + l[i];
    END_REPEAT;
    RETURN(s);
  END_FUNCTION; -- list_to_set

  FUNCTION make_array_of_array(lis: LIST [1:?] OF LIST [1:?] OF GENERIC:t;
                 low1, u1, low2, u2: INTEGER): ARRAY OF ARRAY OF GENERIC:t;
    LOCAL
      res : ARRAY [low1:u1] OF ARRAY [low2:u2] OF GENERIC:t;
    END_LOCAL;
    IF ((u1 - low1) + 1) <> SIZEOF(lis) THEN RETURN(?);
    END_IF;
    IF ((u2 - low2) + 1) <> SIZEOF(lis[1]) THEN RETURN(?);
    END_IF;
```

```
    res := [list_to_array(lis[1],low2,u2),(u1 - low1) + 1];
    REPEAT i := 2 TO HIINDEX(lis) BY 1;
      IF ((u2 - low2) + 1) <> SIZEOF(lis[i]) THEN RETURN(?);
      END_IF;
      res[(low1 + i) - 1] := list_to_array(lis[i],low2,u2);
    END_REPEAT;
    RETURN(res);
  END_FUNCTION; -- make_array_of_array

  FUNCTION mixed_loop_type_set(l: SET [0:?] OF loop): LOGICAL;
    LOCAL
      poly_loop_type : LOGICAL;
    END_LOCAL;
    IF SIZEOF(l) <= 1 THEN RETURN(FALSE);
    END_IF;
    poly_loop_type := 'CONFIG_CONTROL_DESIGN.POLY_LOOP' IN TYPEOF(l[1]);
    REPEAT i := 2 TO SIZEOF(l) BY 1;
      IF ('CONFIG_CONTROL_DESIGN.POLY_LOOP' IN TYPEOF(l[i])) <>
          poly_loop_type THEN RETURN(TRUE);
      END_IF;
    END_REPEAT;
    RETURN(FALSE);
  END_FUNCTION; -- mixed_loop_type_set

  FUNCTION msb_shells(brep: manifold_solid_brep
      ): SET [1:?] OF closed_shell;
    IF SIZEOF(QUERY ( msbtype <* TYPEOF(brep) | (msbtype LIKE
        '*BREP_WITH_VOIDS') )) >= 1 THEN
      RETURN(brep\brep_with_voids.voids + brep.outer);
    ELSE
      RETURN([brep.outer]);
    END_IF;
  END_FUNCTION; -- msb_shells

  FUNCTION msf_curve_check(cv: curve): BOOLEAN;
    IF SIZEOF(['CONFIG_CONTROL_DESIGN.BOUNDED_CURVE',
        'CONFIG_CONTROL_DESIGN.CONIC',
        'CONFIG_CONTROL_DESIGN.CURVE_REPLICA',
        'CONFIG_CONTROL_DESIGN.LINE',
        'CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D'] * TYPEOF(cv)) > 1 THEN
      RETURN(FALSE);
    ELSE
```

```
IF (('CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE' IN TYPEOF(cv)) AND (cv\
    b_spline_curve.self_intersect = FALSE)) OR (cv\b_spline_curve.
    self_intersect = UNKNOWN) THEN RETURN(TRUE);
ELSE
  IF SIZEOF(['CONFIG_CONTROL_DESIGN.CONIC',
      'CONFIG_CONTROL_DESIGN.LINE'] * TYPEOF(cv)) = 1 THEN
    RETURN(TRUE);
  ELSE
    IF 'CONFIG_CONTROL_DESIGN.CURVE_REPLICA' IN TYPEOF(cv) THEN
      RETURN(msf_curve_check(cv\curve_replica.parent_curve));
    ELSE
      IF ('CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D' IN TYPEOF(cv)) AND
           ((cv\offset_curve_3d.self_intersect = FALSE) OR (cv\
          offset_curve_3d.self_intersect = UNKNOWN)) AND (NOT (
          'CONFIG_CONTROL_DESIGN.POLYLINE'
          IN TYPEOF(cv.basis_curve))) THEN
        RETURN(msf_curve_check(cv\offset_curve_3d.basis_curve));
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(cv) THEN
          RETURN(msf_curve_check(cv\pcurve.reference_to_curve\
              representation.items[1]) AND msf_surface_check(cv\
              pcurve.basis_surface));
        ELSE
          IF 'CONFIG_CONTROL_DESIGN.SURFACE_CURVE' IN TYPEOF(cv)
               THEN
            IF msf_curve_check(cv\surface_curve.curve_3d) THEN
              REPEAT i := 1 TO SIZEOF(cv\surface_curve.
                  associated_geometry) BY 1;
                IF 'CONFIG_CONTROL_DESIGN.SURFACE' IN TYPEOF(cv\
                    surface_curve.associated_geometry[i]) THEN
                  IF NOT msf_surface_check(cv\surface_curve.
                      associated_geometry[i]) THEN
                    RETURN(FALSE);
                  END_IF;
                ELSE
                  IF 'CONFIG_CONTROL_DESIGN.PCURVE' IN TYPEOF(cv\
                      surface_curve.associated_geometry[i]) THEN
                    IF NOT msf_curve_check(cv\surface_curve.
                        associated_geometry[i]) THEN
                      RETURN(FALSE);
                    END_IF;
                  END_IF;
```

```
                    END_IF;
                  END_REPEAT;
                  RETURN(TRUE);
                END_IF;
              ELSE
                IF 'CONFIG_CONTROL_DESIGN.POLYLINE' IN TYPEOF(cv) THEN
                  IF SIZEOF(cv\polyline.points) >= 3 THEN
                    RETURN(TRUE);
                  END_IF;
                END_IF;
              END_IF;
            END_IF;
          END_IF;
        END_IF;
      END_IF;
    END_IF;
    RETURN(FALSE);
  END_FUNCTION; -- msf_curve_check

  FUNCTION msf_surface_check(surf: surface): BOOLEAN;
    IF 'CONFIG_CONTROL_DESIGN.ELEMENTARY_SURFACE' IN TYPEOF(surf) THEN
      RETURN(TRUE);
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.SWEPT_SURFACE' IN TYPEOF(surf) THEN
        RETURN(msf_curve_check(surf\swept_surface.swept_curve));
      ELSE
        IF (('CONFIG_CONTROL_DESIGN.OFFSET_SURFACE' IN TYPEOF(surf)) AND (
            surf\offset_surface.self_intersect = FALSE)) OR (surf\
            offset_surface.self_intersect = UNKNOWN) THEN
          RETURN(msf_surface_check(surf\offset_surface.basis_surface));
        ELSE
          IF 'CONFIG_CONTROL_DESIGN.SURFACE_REPLICA' IN TYPEOF(surf) THEN
            RETURN(msf_surface_check(surf\surface_replica.parent_surface));
          ELSE
            IF (('CONFIG_CONTROL_DESIGN.B_SPLINE_SURFACE' IN TYPEOF(surf))
                AND (surf\b_spline_surface.self_intersect = FALSE)) OR (
                surf\b_spline_surface.self_intersect = UNKNOWN) THEN
              RETURN(TRUE);
            END_IF;
          END_IF;
        END_IF;
```

```
      END_IF;
    END_IF;
    RETURN(FALSE);
  END_FUNCTION; -- msf_surface_check

  FUNCTION normalise(arg: vector_or_direction): vector_or_direction;
    LOCAL
      ndim   : INTEGER;
      v      : direction;
      vec    : vector;
      mag    : REAL;
      result : vector_or_direction;
    END_LOCAL;
    IF NOT EXISTS(arg) THEN result := ?;
    ELSE
      ndim := arg.dim;
      IF 'CONFIG_CONTROL_DESIGN.VECTOR' IN TYPEOF(arg) THEN
        BEGIN
          v := dummy_gri || direction(arg.orientation.direction_ratios);
          IF arg.magnitude = 0 THEN RETURN(?);
          ELSE
            vec := dummy_gri || vector(v,1);
          END_IF;
        END;
      ELSE
        v := dummy_gri || direction(arg.direction_ratios);
      END_IF;
      mag := 0;
      REPEAT i := 1 TO ndim BY 1;
        mag := mag + (v.direction_ratios[i] * v.direction_ratios[i]);
      END_REPEAT;
      IF mag > 0 THEN
        mag := SQRT(mag);
        REPEAT i := 1 TO ndim BY 1;
          v.direction_ratios[i] := v.direction_ratios[i] / mag;
        END_REPEAT;
        IF 'CONFIG_CONTROL_DESIGN.VECTOR' IN TYPEOF(arg) THEN
          vec.orientation := v;
          result := vec;
        ELSE
          result := v;
        END_IF;
```

```
    ELSE
      RETURN(?);
    END_IF;
  END_IF;
  RETURN(result);
END_FUNCTION; -- normalise

FUNCTION open_shell_reversed(a_shell: open_shell): oriented_open_shell;
  LOCAL
    the_reverse : oriented_open_shell;
  END_LOCAL;
  IF 'CONFIG_CONTROL_DESIGN.ORIENTED_OPEN_SHELL' IN TYPEOF(a_shell)
       THEN
    the_reverse := dummy_tri || connected_face_set(a_shell\
        connected_face_set.cfs_faces) || open_shell() ||
        oriented_open_shell(a_shell\oriented_open_shell.
        open_shell_element,NOT a_shell\oriented_open_shell.orientation);
  ELSE
    the_reverse := dummy_tri || connected_face_set(a_shell\
        connected_face_set.cfs_faces) || open_shell() ||
        oriented_open_shell(a_shell,FALSE);
  END_IF;
  RETURN(the_reverse);
END_FUNCTION; -- open_shell_reversed

FUNCTION orthogonal_complement(vec: direction): direction;
  LOCAL
    result : direction;
  END_LOCAL;
  IF (vec.dim <> 2) OR (NOT EXISTS(vec)) THEN RETURN(?);
  ELSE
    result := dummy_gri || direction([-vec.direction_ratios[2],vec.
        direction_ratios[1]]);
    RETURN(result);
  END_IF;
END_FUNCTION; -- orthogonal_complement

FUNCTION path_head_to_tail(a_path: path): LOGICAL;
  LOCAL
    n : INTEGER;
    p : BOOLEAN := TRUE;
  END_LOCAL;
```

```
    n := SIZEOF(a_path.edge_list);
    REPEAT i := 2 TO n BY 1;
      p := p AND (a_path.edge_list[i - 1].edge_end :=: a_path.edge_list[i]
          .edge_start);
    END_REPEAT;
    RETURN(p);
  END_FUNCTION; -- path_head_to_tail

  FUNCTION path_reversed(a_path: path): oriented_path;
    LOCAL
      the_reverse : oriented_path;
    END_LOCAL;
    IF 'CONFIG_CONTROL_DESIGN.ORIENTED_PATH' IN TYPEOF(a_path) THEN
      the_reverse := dummy_tri || path(list_of_topology_reversed(a_path.
          edge_list)) || oriented_path(a_path\oriented_path.path_element,
          NOT a_path\oriented_path.orientation);
    ELSE
      the_reverse := dummy_tri || path(list_of_topology_reversed(a_path.
          edge_list)) || oriented_path(a_path,FALSE);
    END_IF;
    RETURN(the_reverse);
  END_FUNCTION; -- path_reversed

  FUNCTION scalar_times_vector(scalar: REAL;
               vec: vector_or_direction): vector;
    LOCAL
      v      : direction;
      mag    : REAL;
      result : vector;
    END_LOCAL;
    IF (NOT EXISTS(scalar)) OR (NOT EXISTS(vec)) THEN RETURN(?);
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.VECTOR' IN TYPEOF(vec) THEN
        v := dummy_gri || direction(vec.orientation.direction_ratios);
        mag := scalar * vec.magnitude;
      ELSE
        v := dummy_gri || direction(vec.direction_ratios);
        mag := scalar;
      END_IF;
      IF mag < 0 THEN
        REPEAT i := 1 TO SIZEOF(v.direction_ratios) BY 1;
          v.direction_ratios[i] := -v.direction_ratios[i];
```

```
      END_REPEAT;
      mag := -mag;
    END_IF;
    result := dummy_gri || vector(normalise(v),mag);
  END_IF;
  RETURN(result);
END_FUNCTION; -- scalar_times_vector

FUNCTION second_proj_axis(z_axis, x_axis, arg: direction
    ): direction;
  LOCAL
    temp   : vector;
    v      : direction;
    y_axis : vector;
  END_LOCAL;
  IF NOT EXISTS(arg) THEN
    v := dummy_gri || direction([0,1,0]);
  ELSE
    v := arg;
  END_IF;
  temp := scalar_times_vector(dot_product(v,z_axis),z_axis);
  y_axis := vector_difference(v,temp);
  temp := scalar_times_vector(dot_product(v,x_axis),x_axis);
  y_axis := vector_difference(y_axis,temp);
  y_axis := normalise(y_axis);
  RETURN(y_axis.orientation);
END_FUNCTION; -- second_proj_axis

FUNCTION set_of_topology_reversed(a_set: set_of_reversible_topology_item
    ): set_of_reversible_topology_item;
  LOCAL
    the_reverse : set_of_reversible_topology_item;
  END_LOCAL;
  the_reverse := [];
  REPEAT i := 1 TO SIZEOF(a_set) BY 1;
    the_reverse := the_reverse + topology_reversed(a_set[i]);
  END_REPEAT;
  RETURN(the_reverse);
END_FUNCTION; -- set_of_topology_reversed

FUNCTION shell_reversed(a_shell: shell): shell;
  IF 'CONFIG_CONTROL_DESIGN.OPEN_SHELL' IN TYPEOF(a_shell) THEN
```

```
      RETURN(open_shell_reversed(a_shell));
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.CLOSED_SHELL' IN TYPEOF(a_shell) THEN
        RETURN(closed_shell_reversed(a_shell));
      ELSE
        RETURN(?);
      END_IF;
    END_IF;
  END_FUNCTION; -- shell_reversed

  FUNCTION surface_weights_positive(b: rational_b_spline_surface
      ): BOOLEAN;
    LOCAL
      result : BOOLEAN := TRUE;
    END_LOCAL;
    REPEAT i := 0 TO b.u_upper BY 1;
      REPEAT j := 0 TO b.v_upper BY 1;
        IF b.weights[i][j] <= 0 THEN result := FALSE;
          RETURN(result);
        END_IF;
      END_REPEAT;
    END_REPEAT;
    RETURN(result);
  END_FUNCTION; -- surface_weights_positive

  FUNCTION topology_reversed(an_item: reversible_topology
      ): reversible_topology;
    IF 'CONFIG_CONTROL_DESIGN.EDGE' IN TYPEOF(an_item) THEN
      RETURN(edge_reversed(an_item));
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.PATH' IN TYPEOF(an_item) THEN
      RETURN(path_reversed(an_item));
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.FACE_BOUND' IN TYPEOF(an_item) THEN
      RETURN(face_bound_reversed(an_item));
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.FACE' IN TYPEOF(an_item) THEN
      RETURN(face_reversed(an_item));
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.SHELL' IN TYPEOF(an_item) THEN
      RETURN(shell_reversed(an_item));
    END_IF;
```

```
    IF 'SET' IN TYPEOF(an_item) THEN
      RETURN(set_of_topology_reversed(an_item));
    END_IF;
    IF 'LIST' IN TYPEOF(an_item) THEN
      RETURN(list_of_topology_reversed(an_item));
    END_IF;
    RETURN(?);
  END_FUNCTION; -- topology_reversed

  FUNCTION unique_version_change_order(c: action): BOOLEAN;
    LOCAL
      ords     : action_directive := c\directed_action.directive;
      assign   : SET OF change_request := [];
      versions : SET OF product_definition_formation := [];
    END_LOCAL;
    REPEAT i := 1 TO SIZEOF(ords.requests) BY 1;
      assign := assign + QUERY ( ara <* bag_to_set(USEDIN(ords.requests[i],
          'CONFIG_CONTROL_DESIGN.ACTION_REQUEST_ASSIGNMENT.' +
          'ASSIGNED_ACTION_REQUEST')) | (
          'CONFIG_CONTROL_DESIGN.CHANGE_REQUEST' IN TYPEOF(ara)) );
    END_REPEAT;
    REPEAT k := 1 TO SIZEOF(assign) BY 1;
      versions := versions + assign[k].items;
    END_REPEAT;
    RETURN(SIZEOF(QUERY ( vers <* versions | (NOT (SIZEOF(
        QUERY ( other_vers <* (versions - vers) | (vers.of_product :=:
        other_vers.of_product) )) = 0)) )) = 0);
  END_FUNCTION; -- unique_version_change_order

  FUNCTION using_items(item: founded_item_select;
               checked_items: SET OF founded_item_select
      ): SET OF founded_item_select;
    LOCAL
      next_items      : SET OF founded_item_select;
      new_check_items : SET OF founded_item_select;
      result_items    : SET OF founded_item_select;
    END_LOCAL;
    result_items := [];
    new_check_items := checked_items + item;
    next_items := QUERY ( z <* bag_to_set(USEDIN(item,'')) | ((
        'CONFIG_CONTROL_DESIGN.REPRESENTATION_ITEM' IN TYPEOF(z)) OR (
        'CONFIG_CONTROL_DESIGN.FOUNDED_ITEM' IN TYPEOF(z))) );
```

```
    IF SIZEOF(next_items) > 0 THEN
      REPEAT i := 1 TO HIINDEX(next_items) BY 1;
        IF NOT (next_items[i] IN new_check_items) THEN
          result_items := result_items + next_items[i] + using_items(
              next_items[i],new_check_items);
        END_IF;
      END_REPEAT;
    END_IF;
    RETURN(result_items);
  END_FUNCTION; -- using_items

  FUNCTION using_representations(item: founded_item_select
      ): SET OF representation;
    LOCAL
      results            : SET OF representation;
      intermediate_items : SET OF founded_item_select;
      result_bag         : BAG OF representation;
    END_LOCAL;
    results := [];
    result_bag := USEDIN(item,
    'CONFIG_CONTROL_DESIGN.REPRESENTATION.ITEMS');
    IF SIZEOF(result_bag) > 0 THEN
      REPEAT i := 1 TO HIINDEX(result_bag) BY 1;
        results := results + result_bag[i];
      END_REPEAT;
    END_IF;
    intermediate_items := using_items(item,[]);
    IF SIZEOF(intermediate_items) > 0 THEN
      REPEAT i := 1 TO HIINDEX(intermediate_items) BY 1;
        result_bag := USEDIN(intermediate_items[i],
            'CONFIG_CONTROL_DESIGN.REPRESENTATION.ITEMS');
        IF SIZEOF(result_bag) > 0 THEN
          REPEAT j := 1 TO HIINDEX(result_bag) BY 1;
            results := results + result_bag[j];
          END_REPEAT;
        END_IF;
      END_REPEAT;
    END_IF;
    RETURN(results);
  END_FUNCTION; -- using_representations
```

```
FUNCTION valid_calendar_date(date: calendar_date): LOGICAL;
  IF NOT ((1 <= date.day_component) AND (date.day_component <= 31))
       THEN RETURN(FALSE);
  END_IF;
  CASE date.month_component OF
    4      :       RETURN((1 <= date.day_component) AND (date.
           day_component <= 30));
    6      :       RETURN((1 <= date.day_component) AND (date.
           day_component <= 30));
    9      :       RETURN((1 <= date.day_component) AND (date.
           day_component <= 30));
    11     :       RETURN((1 <= date.day_component) AND (date.
           day_component <= 30));
    2      :       BEGIN
        IF leap_year(date.year_component) THEN
          RETURN((1 <= date.day_component) AND
          (date.day_component <= 29));
        ELSE
          RETURN((1 <= date.day_component) AND
          (date.day_component <= 28));
        END_IF;
      END;
    OTHERWISE  :         RETURN(TRUE);
    END_CASE;
END_FUNCTION; -- valid_calendar_date

FUNCTION valid_geometrically_bounded_wf_curve(crv: curve): BOOLEAN;
  IF SIZEOF(['CONFIG_CONTROL_DESIGN.POLYLINE',
      'CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE',
      'CONFIG_CONTROL_DESIGN.ELLIPSE','CONFIG_CONTROL_DESIGN.CIRCLE'] *
      TYPEOF(crv)) = 1 THEN RETURN(TRUE);
  ELSE
    IF 'CONFIG_CONTROL_DESIGN.TRIMMED_CURVE' IN TYPEOF(crv) THEN
      IF SIZEOF(['CONFIG_CONTROL_DESIGN.LINE',
          'CONFIG_CONTROL_DESIGN.PARABOLA',
          'CONFIG_CONTROL_DESIGN.HYPERBOLA'] * TYPEOF(crv\trimmed_curve.
          basis_curve)) = 1 THEN RETURN(TRUE);
      ELSE
        RETURN(valid_geometrically_bounded_wf_curve(crv\trimmed_curve.
            basis_curve));
      END_IF;
    ELSE
```

```
      IF 'CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D' IN TYPEOF(crv) THEN
        RETURN(valid_geometrically_bounded_wf_curve(crv\offset_curve_3d.
            basis_curve));
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.CURVE_REPLICA' IN TYPEOF(crv) THEN
          RETURN(valid_geometrically_bounded_wf_curve(crv\curve_replica.
              parent_curve));
        ELSE
          IF 'CONFIG_CONTROL_DESIGN.COMPOSITE_CURVE' IN TYPEOF(crv)
               THEN
            RETURN(SIZEOF(QUERY ( ccs <* crv\composite_curve.segments |
                (NOT valid_geometrically_bounded_wf_curve(ccs.
                parent_curve)) )) = 0);
          END_IF;
        END_IF;
      END_IF;
    END_IF;
  END_IF;
  RETURN(FALSE);
END_FUNCTION; -- valid_geometrically_bounded_wf_curve

FUNCTION valid_geometrically_bounded_wf_point(pnt: point): BOOLEAN;
  IF 'CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN TYPEOF(pnt) THEN
    RETURN(TRUE);
  ELSE
    IF 'CONFIG_CONTROL_DESIGN.POINT_ON_CURVE' IN TYPEOF(pnt) THEN
      RETURN(valid_geometrically_bounded_wf_curve(pnt\point_on_curve.
          basis_curve));
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.POINT_REPLICA' IN TYPEOF(pnt) THEN
        RETURN(valid_geometrically_bounded_wf_point(pnt\point_replica.
            parent_pt));
      END_IF;
    END_IF;
  END_IF;
  RETURN(FALSE);
END_FUNCTION; -- valid_geometrically_bounded_wf_point

FUNCTION valid_measure_value(m: measure_value): BOOLEAN;
  IF 'REAL' IN TYPEOF(m) THEN
    RETURN(m > 0);
  ELSE
```

```
    IF 'INTEGER' IN TYPEOF(m) THEN RETURN(m > 0);
    ELSE
      RETURN(TRUE);
    END_IF;
  END_IF;
END_FUNCTION; -- valid_measure_value

FUNCTION valid_time(time: local_time): BOOLEAN;
  IF EXISTS(time.second_component) THEN
    RETURN(EXISTS(time.minute_component));
  ELSE
    RETURN(TRUE);
  END_IF;
END_FUNCTION; -- valid_time

FUNCTION valid_units(m: measure_with_unit): BOOLEAN;
  IF 'CONFIG_CONTROL_DESIGN.LENGTH_MEASURE' IN TYPEOF(m.value_component)
       THEN
    IF derive_dimensional_exponents(m.unit_component) <>
        dimensional_exponents(1,0,0,0,0,0,0) THEN RETURN(FALSE);
    END_IF;
  END_IF;
  IF 'CONFIG_CONTROL_DESIGN.MASS_MEASURE' IN TYPEOF(m.value_component)
       THEN
    IF derive_dimensional_exponents(m.unit_component) <>
        dimensional_exponents(0,1,0,0,0,0,0) THEN
      RETURN(FALSE);
    END_IF;
  END_IF;
  IF 'CONFIG_CONTROL_DESIGN.TIME_MEASURE' IN TYPEOF(m.value_component)
       THEN
    IF derive_dimensional_exponents(m.unit_component) <>
        dimensional_exponents(0,0,1,0,0,0,0) THEN RETURN(FALSE);
    END_IF;
  END_IF;
  IF 'CONFIG_CONTROL_DESIGN.ELECTRIC_CURRENT_MEASURE' IN TYPEOF(m.
      value_component) THEN
    IF derive_dimensional_exponents(m.unit_component) <>
        dimensional_exponents(0,0,0,1,0,0,0) THEN RETURN(FALSE);
    END_IF;
  END_IF;
  IF 'CONFIG_CONTROL_DESIGN.THERMODYNAMIC_TEMPERATURE_MEASURE' IN
```

```
    TYPEOF(m.value_component) THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0,0,0,0,1,0,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.AMOUNT_OF_SUBSTANCE_MEASURE' IN TYPEOF(m.
    value_component) THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0,0,0,0,0,1,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.LUMINOUS_INTENSITY_MEASURE' IN TYPEOF(m.
    value_component) THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0,0,0,0,0,0,1) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.PLANE_ANGLE_MEASURE' IN TYPEOF(m.
    value_component) THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0,0,0,0,0,0,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.SOLID_ANGLE_MEASURE' IN TYPEOF(m.
    value_component) THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(0,0,0,0,0,0,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.AREA_MEASURE' IN TYPEOF(m.value_component)
     THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(2,0,0,0,0,0,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.VOLUME_MEASURE' IN TYPEOF(m.value_component)
     THEN
  IF derive_dimensional_exponents(m.unit_component) <>
      dimensional_exponents(3,0,0,0,0,0,0) THEN RETURN(FALSE);
  END_IF;
END_IF;
IF 'CONFIG_CONTROL_DESIGN.RATIO_MEASURE' IN TYPEOF(m.value_component)
```

```
        THEN
      IF derive_dimensional_exponents(m.unit_component) <>
          dimensional_exponents(0,0,0,0,0,0,0) THEN RETURN(FALSE);
      END_IF;
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.POSITIVE_LENGTH_MEASURE' IN TYPEOF(m.
        value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
          dimensional_exponents(1,0,0,0,0,0,0) THEN RETURN(FALSE);
      END_IF;
    END_IF;
    IF 'CONFIG_CONTROL_DESIGN.POSITIVE_PLANE_ANGLE_MEASURE' IN TYPEOF(m.
        value_component) THEN
      IF derive_dimensional_exponents(m.unit_component) <>
          dimensional_exponents(0,0,0,0,0,0,0) THEN RETURN(FALSE);
      END_IF;
    END_IF;
    RETURN(TRUE);
  END_FUNCTION; -- valid_units

  FUNCTION valid_wireframe_edge_curve(crv: curve): BOOLEAN;
    IF SIZEOF(['CONFIG_CONTROL_DESIGN.LINE','CONFIG_CONTROL_DESIGN.CONIC',
        'CONFIG_CONTROL_DESIGN.B_SPLINE_CURVE',
        'CONFIG_CONTROL_DESIGN.POLYLINE'] * TYPEOF(crv)) = 1 THEN
      RETURN(TRUE);
    ELSE
      IF 'CONFIG_CONTROL_DESIGN.CURVE_REPLICA' IN TYPEOF(crv) THEN
        RETURN(valid_wireframe_edge_curve(crv\curve_replica.parent_curve));
      ELSE
        IF 'CONFIG_CONTROL_DESIGN.OFFSET_CURVE_3D' IN TYPEOF(crv) THEN
          RETURN(valid_wireframe_edge_curve(crv\offset_curve_3d.
              basis_curve));
        END_IF;
      END_IF;
    END_IF;
    RETURN(FALSE);
  END_FUNCTION; -- valid_wireframe_edge_curve

  FUNCTION valid_wireframe_vertex_point(pnt: point): BOOLEAN;
    IF 'CONFIG_CONTROL_DESIGN.CARTESIAN_POINT' IN TYPEOF(pnt) THEN
      RETURN(TRUE);
    ELSE
```

```
      IF 'CONFIG_CONTROL_DESIGN.POINT_REPLICA' IN TYPEOF(pnt) THEN
        RETURN(valid_wireframe_vertex_point(pnt\point_replica.parent_pt));
      END_IF;
    END_IF;
    RETURN(FALSE);
  END_FUNCTION; -- valid_wireframe_vertex_point

  FUNCTION vector_difference(arg1, arg2: vector_or_direction
      ): vector;
    LOCAL
      ndim   : INTEGER;
      mag2   : REAL;
      mag1   : REAL;
      mag    : REAL;
      res    : direction;
      vec1   : direction;
      vec2   : direction;
      result : vector;
    END_LOCAL;
    IF (NOT EXISTS(arg1)) OR (NOT EXISTS(arg2)) OR (arg1.dim <> arg2.dim)
         THEN RETURN(?);
    ELSE
      BEGIN
        IF 'CONFIG_CONTROL_DESIGN.VECTOR' IN TYPEOF(arg1) THEN
          mag1 := arg1.magnitude;
          vec1 := arg1.orientation;
        ELSE
          mag1 := 1;
          vec1 := arg1;
        END_IF;
        IF 'CONFIG_CONTROL_DESIGN.VECTOR' IN TYPEOF(arg2) THEN
          mag2 := arg2.magnitude;
          vec2 := arg2.orientation;
        ELSE
          mag2 := 1;
          vec2 := arg2;
        END_IF;
        vec1 := normalise(vec1);
        vec2 := normalise(vec2);
        ndim := SIZEOF(vec1.direction_ratios);
        mag := 0;
        res := dummy_gri || direction(vec1.direction_ratios);
```

```
        REPEAT i := 1 TO ndim BY 1;
          res.direction_ratios[i] := (mag1 * vec1.direction_ratios[i]) + (
              mag2 * vec2.direction_ratios[i]);
          mag := mag + (res.direction_ratios[i] * res.direction_ratios[i]);
        END_REPEAT;
        IF mag > 0 THEN
          result := dummy_gri || vector(res,SQRT(mag));
        ELSE
          result := dummy_gri || vector(vec1,0);
        END_IF;
      END;
    END_IF;
    RETURN(result);
  END_FUNCTION; -- vector_difference

END_SCHEMA; -- config_control_design
(*
```

This Page Intentionally Left Blank

This Page Intentionally Left Blank

This Page Intentionally Left Blank

Annex L
(informative)

Bibliography

The following are informative references:

[1] *Guidelines for the Development and Approval of STEP Application Protocols, Version 1.1*, ISO TC184/SC4/WG4 N66, January 1993.

[2] Marks' Standard Handbook for Mechanical Engineers, edited by Avallone and Baumeister, Ninth Edition, McGraw-Hill, 1987.

[3] *IDEF0 (ICAM Definition Language 0)*, Federal Information Processing Standards Publication 183, Integration Definition for Function Modeling (IDEF0), FIPS PUB 183, National Institute of Standards and Technology, December 1993.

[4] *IDEF1X (ICAM Definition Language 1 Extended)*, Federal Information Processing Standards Publication 184, Integration Definition for Information Modeling (IDEF1X), FIPS PUB 184, National Institute of Standards and Technology, December 1993.

Index

action
AIM EXPRESS expanded listing entities 280
AIM EXPRESS short listing imported entity modifications 125
action_assignment
AIM EXPRESS expanded listing entities 280
action_directive
AIM EXPRESS expanded listing entities 280
AIM EXPRESS short listing imported entity modifications 126
action_method
AIM EXPRESS expanded listing entities 280
mapping_table 53-55
action_request_assignment
AIM EXPRESS expanded listing entities 280
action_request_solution
AIM EXPRESS expanded listing entities 280
AIM EXPRESS short listing imported entity modifications 120
action_request_status
AIM EXPRESS expanded listing entities 281
AIM EXPRESS short listing imported entity modifications 119
mapping_table 58
action_status
AIM EXPRESS expanded listing entities 281
acu_requires_security_classification
AIM EXPRESS expanded listing rules 338
AIM EXPRESS short listing rules 186
acyclic_curve_replica
AIM EXPRESS expanded listing functions 354
acyclic_mapped_representation
AIM EXPRESS expanded listing functions 354
acyclic_point_replica
AIM EXPRESS expanded listing functions 355
acyclic_product_category_relationship
AIM EXPRESS expanded listing functions 355
acyclic_product_definition_relationship
AIM EXPRESS expanded listing functions 356
acyclic_surface_replica
AIM EXPRESS expanded listing functions 357
additional_data
application object attribute 32
mapping table 56

additional_design_information
application assertion 33, 34
application object 15
mapping table 61, 73
address
AIM EXPRESS expanded listing entities 281
application object attribute 25
mapping table 44
adopted_solution
application object attribute 16
mapping table 53
advanced_b_rep
application object 15
mapping table 41
advanced_boundary_representation
unit of functionality 10
advanced_brep_representation
mapping table 41
advanced_brep_shape_representation
AIM EXPRESS expanded listing entities 281
advanced_face
AIM EXPRESS expanded listing entities 282
ahead_or_behind
AIM EXPRESS expanded listing types 270
alternate_part
application assertion 35
application object 15
mapping table 47, 75
alternate_product_relationship
AIM EXPRESS expanded listing entities 284
mapping table 47, 75
analysis_data
application object attribute 32
mapping table 56
application_context
AIM EXPRESS expanded listing entities 284
AIM EXPRESS short listing imported entity modifications 116
application_context_element
AIM EXPRESS expanded listing entities 284
application_context_requires_ap_definition
AIM EXPRESS expanded listing rules 338
AIM EXPRESS short listing rules 131

application_protocol_definition
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 116
approval
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 121
application assertion 33-38
application object 16
mapping table 42, 43, 56, 59, 64, 67, 73, 78, 85
approval_assignment
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 121
approval_date_time
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 122
approval_date_time_constraints
AIM EXPRESS expanded listing rules 339
AIM EXPRESS short listing rules 192
approval_item
mapping table 56, 59
approval_person_organization
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 121
mapping table 43
approval_person_organization_constraints
AIM EXPRESS expanded listing rules 339
AIM EXPRESS short listing rules 193
approval_relationship
AIM EXPRESS expanded listing entities 285
approval_requires_approval_date_time
AIM EXPRESS expanded listing rules 339
AIM EXPRESS short listing rules 157
approval_requires_approval_person_organization
AIM EXPRESS expanded listing rules 339
AIM EXPRESS short listing rules 157
approval_role
AIM EXPRESS expanded listing entities 285
approval_status
AIM EXPRESS expanded listing entities 285
AIM EXPRESS short listing imported entity modifications 120
mapping table 42, 77
approvals_are_assigned

AIM EXPRESS expanded listing rules 339
AIM EXPRESS short listing rules 156
approved_item
AIM EXPRESS expanded listing types 270
AIM EXPRESS short listing types 102
mapping table 64, 67, 73, 78, 85
area_measure
AIM EXPRESS expanded listing types 270
area_measure_with_unit
AIM EXPRESS expanded listing entities 286
area_unit
AIM EXPRESS expanded listing entities 286
as_required
application object attribute 20
mapping table 49
as_required_quantity
AIM EXPRESS expanded listing rules 340
AIM EXPRESS short listing rules 171
assembly_component_usage
AIM EXPRESS expanded listing entities 286
AIM EXPRESS short listing imported entity modifications 130
mapping table 48, 49
assembly_component_usage_substitute
AIM EXPRESS expanded listing entities 286
mapping table 51
assembly_shape_is_defined
AIM EXPRESS expanded listing functions 357
AIM EXPRESS short listing functions 200
associated_surface
AIM EXPRESS expanded listing functions 358
authorization
unit of functionality 11
axis1_placement
AIM EXPRESS expanded listing entities 286
axis2_placement
AIM EXPRESS expanded listing types 270
axis2_placement_2d
AIM EXPRESS expanded listing entities 287
axis2_placement_3d
AIM EXPRESS expanded listing entities 287
b_spline_curve
AIM EXPRESS expanded listing entities 287

b_spline_curve_form
AIM EXPRESS expanded listing types 271
b_spline_curve_with_knots
AIM EXPRESS expanded listing entities 288
b_spline_surface
AIM EXPRESS expanded listing entities 288
b_spline_surface_form
AIM EXPRESS expanded listing types 271
b_spline_surface_with_knots
AIM EXPRESS expanded listing entities 289
bag_to_set
AIM EXPRESS expanded listing functions 359
base_axis
AIM EXPRESS expanded listing functions 359
bezier_curve
AIM EXPRESS expanded listing entities 289
bezier_surface
AIM EXPRESS expanded listing entities 289
bill_of_material
unit of functionality 11
boolean_choose
AIM EXPRESS expanded listing functions 360
boolean_operand
AIM EXPRESS expanded listing types 271
boundary representation model
definition 8
boundary_curve
AIM EXPRESS expanded listing entities 290
bounded_curve
AIM EXPRESS expanded listing entities 290
bounded_surface
AIM EXPRESS expanded listing entities 290
brep_with_voids
AIM EXPRESS expanded listing entities 290
build_2axes
AIM EXPRESS expanded listing functions 360
build_axes
AIM EXPRESS expanded listing functions 360
cad_filename
application object attribute 18
mapping table 72
calendar_date

AIM EXPRESS expanded listing entities 290
cartesian_point
AIM EXPRESS expanded listing entities 291
cartesian_transformation_operator
AIM EXPRESS expanded listing entities 291
cartesian_transformation_operator_3d
AIM EXPRESS expanded listing entities 291
cc_design_approval
AIM EXPRESS expanded listing entities 291
AIM EXPRESS short listing entities 111
mapping table 42
cc_design_certification
AIM EXPRESS expanded listing entities 291
AIM EXPRESS short listing entities 110
mapping table 83
cc_design_contract
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing entities 112
cc_design_date_and_time_assignment
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing entities 114
cc_design_date_time_correlation
AIM EXPRESS expanded listing functions 361
AIM EXPRESS short listing functions 198
cc_design_person_and_organization_assignment
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing entities 113
mapping table 43
cc_design_person_and_organization_correlation
AIM EXPRESS expanded listing functions 362
AIM EXPRESS short listing functions 195
cc_design_security_classification
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing entities 112
cc_design_specification_reference
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing entities 115
certification
AIM EXPRESS expanded listing entities 292
AIM EXPRESS short listing imported entity modifications 120
certification_assignment
AIM EXPRESS expanded listing entities 292

certification_required
application object attribute 30
mapping table 83
certification_requires_approval
AIM EXPRESS expanded listing rules 340
AIM EXPRESS short listing rules 154
certification_requires_date_time
AIM EXPRESS expanded listing rules 340
AIM EXPRESS short listing rules 155
certification_type
AIM EXPRESS expanded listing entities 293
AIM EXPRESS short listing imported entity modifications 120
certified_item
AIM EXPRESS expanded listing types 271
AIM EXPRESS short listing types 101
change
AIM EXPRESS expanded listing entities 293
AIM EXPRESS short listing entities 109
mapping table 53
change_date
application object attribute 17
mapping table 54
change_order
application object 16
mapping table 53
change_request
AIM EXPRESS expanded listing entities 293
AIM EXPRESS short listing entities 107
application object 17
mapping table 54
change_request_item
AIM EXPRESS expanded listing types 271
AIM EXPRESS short listing types 101
mapping table 59
change_request_requires_approval
AIM EXPRESS expanded listing rules 340
AIM EXPRESS short listing rules 136
change_request_requires_date_time
AIM EXPRESS expanded listing rules 340
AIM EXPRESS short listing rules 137
change_request_requires_person_organization
AIM EXPRESS expanded listing rules 341

AIM EXPRESS short listing rules 136
change_requires_approval
AIM EXPRESS expanded listing rules 341
AIM EXPRESS short listing rules 138
change_requires_date_time
AIM EXPRESS expanded listing rules 341
AIM EXPRESS short listing rules 139
characterized_definition
AIM EXPRESS expanded listing types 271
characterized_product_definition
AIM EXPRESS expanded listing types 271
circle
AIM EXPRESS expanded listing entities 293
classified_item
AIM EXPRESS expanded listing types 272
AIM EXPRESS short listing types 103
closed_shell
AIM EXPRESS expanded listing entities 293
closed_shell_reversed
AIM EXPRESS expanded listing functions 364
compatible_dimension
AIM EXPRESS expanded listing rules 341
component_assembly_position
application assertion 35
application object 17
mapping table 47, 50, 51
component_quantity
application object attribute 20, 27
mapping table 49, 65
composite_curve
AIM EXPRESS expanded listing entities 293
composite_curve_on_surface
AIM EXPRESS expanded listing entities 293
composite_curve_segment
AIM EXPRESS expanded listing entities 294
conditional_reverse
AIM EXPRESS expanded listing functions 365
configuration_design
AIM EXPRESS expanded listing entities 294
mapping table 67
configuration_effectivity
AIM EXPRESS expanded listing entities 294

configuration_item
AIM EXPRESS expanded listing entities 294
AIM EXPRESS short listing imported entity modifications 129
mapping table 67, 68
configuration_item_requires_approval
AIM EXPRESS expanded listing rules 341
AIM EXPRESS short listing rules 184
configuration_item_requires_person_organization
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 181
conic
AIM EXPRESS expanded listing entities 295
conical_surface
AIM EXPRESS expanded listing entities 295
connected_edge_set
AIM EXPRESS expanded listing entities 295
connected_face_set
AIM EXPRESS expanded listing entities 295
consequence
application object attribute 17
mapping table 55
constraints_composite_curve_on_surface
AIM EXPRESS expanded listing functions 365
constraints_geometry_shell_based_surface_model
AIM EXPRESS expanded listing functions 365
constraints_geometry_shell_based_wireframe_model
AIM EXPRESS expanded listing functions 366
constraints_param_b_spline
AIM EXPRESS expanded listing functions 366
constraints_rectangular_composite_surface
AIM EXPRESS expanded listing functions 367
context_dependent_measure
AIM EXPRESS expanded listing types 272
context_dependent_shape_representation
AIM EXPRESS expanded listing entities 295
AIM EXPRESS short listing imported entity modifications 127
mapping table 47, 50
context_dependent_unit
AIM EXPRESS expanded listing entities 295
contract
AIM EXPRESS expanded listing entities 296
AIM EXPRESS short listing imported entity modifications 122

mapping table 76
contract_assignment
AIM EXPRESS expanded listing entities 296
contract_number
application object attribute 24
mapping table 76
contract_requires_approval
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 159
contract_requires_person_organization
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 160
contract_type
AIM EXPRESS expanded listing entities 296
AIM EXPRESS short listing imported entity modifications 122
contracted_item
AIM EXPRESS expanded listing types 272
AIM EXPRESS short listing types 102
conversion_based_unit
AIM EXPRESS expanded listing entities 296
coordinated_assembly_and_shape
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 184
coordinated_universal_time_offset
AIM EXPRESS expanded listing entities 296
count_measure
AIM EXPRESS expanded listing types 272
creation_date
application object attribute 18
mapping table 72
cross_product
AIM EXPRESS expanded listing functions 368
curve
AIM EXPRESS expanded listing entities 296
curve_bounded_surface
AIM EXPRESS expanded listing entities 296
curve_on_surface
AIM EXPRESS expanded listing types 272
curve_replica
AIM EXPRESS expanded listing entities 297
curve_weights_positive
AIM EXPRESS expanded listing functions 369

cylindrical_surface
AIM EXPRESS expanded listing entities 297
date
AIM EXPRESS expanded listing entities 297
AIM EXPRESS short listing imported entity modifications 126
application object attribute 16
mapping table 42, 58
date_and_time
AIM EXPRESS expanded listing entities 297
mapping_table 42, 54, 64, 72
date_and_time_assignment
AIM EXPRESS expanded listing entities 297
date_time_item
AIM EXPRESS expanded listing types 272
AIM EXPRESS short listing types 104
date_time_role
AIM EXPRESS expanded listing entities 298
AIM EXPRESS short listing imported entity modifications 124
date_time_select
AIM EXPRESS expanded listing types 272
dated_effectivity
AIM EXPRESS expanded listing entities 298
mapping table 64
day_in_month_number
AIM EXPRESS expanded listing types 273
day_in_week_number
AIM EXPRESS expanded listing types 273
day_in_year_number
AIM EXPRESS expanded listing types 273
definitional_representation
AIM EXPRESS expanded listing entities 298
degenerate_pcurve
AIM EXPRESS expanded listing entities 298
degenerate_toroidal_surface
AIM EXPRESS expanded listing entities 298
dependent_instantiable_action_directive
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 177
dependent_instantiable_approval_status
AIM EXPRESS expanded listing rules 342
AIM EXPRESS short listing rules 178
dependent_instantiable_certification_type

AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 180
dependent_instantiable_contract_type
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 180
dependent_instantiable_date
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 174
dependent_instantiable_date_time_role
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 176
dependent_instantiable_document_type
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 179
dependent_instantiable_named_unit
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 175
dependent_instantiable_parametric_representation_context
AIM EXPRESS expanded listing rules 343
AIM EXPRESS short listing rules 188
dependent_instantiable_person_and_organization_role
AIM EXPRESS expanded listing rules 344
AIM EXPRESS short listing rules 177
dependent_instantiable_representation_item
AIM EXPRESS expanded listing rules 344
AIM EXPRESS short listing rules 175
dependent_instantiable_security_classification_level
AIM EXPRESS expanded listing rules 344
AIM EXPRESS short listing rules 178
dependent_instantiable_shape_representation
AIM EXPRESS expanded listing rules 344
AIM EXPRESS short listing rules 174
derive_dimensional_exponents
AIM EXPRESS expanded listing functions 369
description
application object attribute 18, 33
mapping table 57, 72
descriptive_measure
AIM EXPRESS expanded listing types 273
design phase
definition 8
design_activity_control

unit of functionality 11
design_context
AIM EXPRESS expanded listing entities 298
AIM EXPRESS short listing entities 105
design_context_for_property
AIM EXPRESS expanded listing rules 344
AIM EXPRESS short listing rules 132
design_discipline_product_definition
application assertion 34-37
application object 18
mapping table 45, 72, 73, 78, 81
design_information
unit of functionality 12
design_make_from_relationship
AIM EXPRESS expanded listing entities 299
AIM EXPRESS short listing entities 106
mapping table 48
design_specification
application object 19
mapping table 61
dimension_count
AIM EXPRESS expanded listing types 273
dimension_of
AIM EXPRESS expanded listing functions 370
dimensional_exponents
AIM EXPRESS expanded listing entities 299
dimensions_for_si_unit
AIM EXPRESS expanded listing functions 370
directed_action
AIM EXPRESS expanded listing entities 299
direction
AIM EXPRESS expanded listing entities 299
discipline_id
application object attribute 19
mapping table 73
document
AIM EXPRESS expanded listing entities 299
mapping_table 61, 62, 72
document_reference
AIM EXPRESS expanded listing entities 299
mapping_table 62
document_relationship

AIM EXPRESS expanded listing entities 300
mapping_table 61
document_to_product_definition
AIM EXPRESS expanded listing rules 345
AIM EXPRESS short listing rules 170
document_type
AIM EXPRESS expanded listing entities 300
AIM EXPRESS short listing imported entity modifications 125
document_usage_constraint
AIM EXPRESS expanded listing entities 300
mapping_table 62
document_with_class
AIM EXPRESS expanded listing entities 300
dot_product
AIM EXPRESS expanded listing functions 371
dummy_gri
AIM EXPRESS expanded listing constants 270
AIM EXPRESS short listing constants 100
dummy_tri
AIM EXPRESS expanded listing constants 270
AIM EXPRESS short listing constants 100
edge
AIM EXPRESS expanded listing entities 300
edge_based_wireframe_model
AIM EXPRESS expanded listing entities 300
edge_based_wireframe_shape_representation
AIM EXPRESS expanded listing entities 300
edge_curve
AIM EXPRESS expanded listing entities 302
edge_loop
AIM EXPRESS expanded listing entities 302
edge_reversed
AIM EXPRESS expanded listing functions 371
effectivity
AIM EXPRESS expanded listing entities 302
AIM EXPRESS short listing imported entity modifications 130
unit of functionality 12
effectivity_requires_approval
AIM EXPRESS expanded listing rules 345
AIM EXPRESS short listing rules 183
element
application object attribute 31

mapping table 62
elementary_surface
AIM EXPRESS expanded listing entities 302
ellipse
AIM EXPRESS expanded listing entities 302
end_date
application object attribute 25
mapping table 64
end_item_identification
unit of functionality 13
engineering_assembly
application assertion 34
application object 19
mapping table 47, 48, 73
engineering_make_from
application object 20
mapping table 48
engineering_next_higher_assembly
application assertion 35
application object 20
mapping table 48, 50, 51
engineering_promissory_usage
application object 21
mapping table 51
evaluated_degenerate_pcurve
AIM EXPRESS expanded listing entities 303
executed_action
AIM EXPRESS expanded listing entities 303
face
AIM EXPRESS expanded listing entities 303
face_bound
AIM EXPRESS expanded listing entities 303
face_bound_reversed
AIM EXPRESS expanded listing functions 372
face_outer_bound
AIM EXPRESS expanded listing entities 303
face_reversed
AIM EXPRESS expanded listing functions 372
face_surface
AIM EXPRESS expanded listing entities 303
facetted_b_rep
application object 21

mapping table 69
faceted_boundary_representation
unit of functionality 13
faceted_brep
AIM EXPRESS expanded listing entities 303
faceted_brep_shape_representation
AIM EXPRESS expanded listing entities 304
mapping table 69
first_proj_axis
AIM EXPRESS expanded listing functions 372
founded_item
AIM EXPRESS expanded listing entities 305
founded_item_select
AIM EXPRESS expanded listing types 273
from_effectivity_id
application object attribute 27
mapping table 65
functionally_defined_transformation
AIM EXPRESS expanded listing entities 305
gbsf_check_curve
AIM EXPRESS expanded listing functions 373
gbsf_check_point
AIM EXPRESS expanded listing functions 375
gbsf_check_surface
AIM EXPRESS expanded listing functions 376
geometric_curve_set
AIM EXPRESS expanded listing entities 305
geometric_model_representation
application assertion 35, 37
application object 21
mapping table 80, 82
geometric_representation_context
AIM EXPRESS expanded listing entities 305
geometric_representation_item
AIM EXPRESS expanded listing entities 305
AIM EXPRESS short listing imported entity modifications 128
geometric_representation_item_3d
AIM EXPRESS expanded listing rules 345
AIM EXPRESS short listing rules 187
geometric_set
AIM EXPRESS expanded listing entities 305
geometric_set_select

AIM EXPRESS expanded listing types 273
geometrically_bounded_surface_shape_representation
AIM EXPRESS expanded listing entities 306
geometrically_bounded_wireframe_shape_representation
AIM EXPRESS expanded listing entities 307
get_basis_surface
AIM EXPRESS expanded listing functions 378
global_uncertainty_assigned_context
AIM EXPRESS expanded listing entities 308
global_unit_assigned_context
AIM EXPRESS expanded listing entities 308
AIM EXPRESS short listing imported entity modifications 125
global_unit_assignment
AIM EXPRESS expanded listing rules 345
AIM EXPRESS short listing rules 171
hour_in_day
AIM EXPRESS expanded listing types 273
hyperbola
AIM EXPRESS expanded listing entities 308
identifier
AIM EXPRESS expanded listing types 273
intersection_curve
AIM EXPRESS expanded listing entities 308
item_defined_transformation
AIM EXPRESS expanded listing entities 308
item_id
application object attribute 28
mapping table 67
item_in_context
AIM EXPRESS expanded listing functions 378
knot_type
AIM EXPRESS expanded listing types 273
label
AIM EXPRESS expanded listing types 274
leap_year
AIM EXPRESS expanded listing functions 379
length_measure
AIM EXPRESS expanded listing types 274
length_measure_with_unit
AIM EXPRESS expanded listing entities 308
length_unit
AIM EXPRESS expanded listing entities 309

line
AIM EXPRESS expanded listing entities 309
list_face_loops
AIM EXPRESS expanded listing functions 379
list_of_reversible_topology_item
AIM EXPRESS expanded listing types 274
list_of_topology_reversed
AIM EXPRESS expanded listing functions 379
list_to_array
AIM EXPRESS expanded listing functions 380
list_to_set
AIM EXPRESS expanded listing functions 380
local_time
AIM EXPRESS expanded listing entities 309
loop
AIM EXPRESS expanded listing entities 309
lot_effectivity
AIM EXPRESS expanded listing entities 309
mapping table 64
lot_number
application object attribute 26
mapping table 64
lot_size
application object attribute 26
mapping table 65
lot_size_unit_of_measure
application object attribute 26
mapping table 65
make_array_of_array
AIM EXPRESS expanded listing functions 380
make_or_buy_code
application object attribute 24
mapping table 77
manifold_solid_brep
AIM EXPRESS expanded listing entities 310
manifold_surface_shape_representation
AIM EXPRESS expanded listing entities 310
manifold_surface_with_topology
application object 22
mapping table 70
unit of functionality 13
manifold_surface_with_topology_representation

mapping table 70
mapped_item
AIM EXPRESS expanded listing entities 314
mapping table 47
mass_measure
AIM EXPRESS expanded listing types 274
mass_measure_with_unit
AIM EXPRESS expanded listing entities 314
mass_unit
AIM EXPRESS expanded listing entities 314
material_specification
application object 22
mapping table 61
measure
mapping table 49, 65, 66
measure_value
AIM EXPRESS expanded listing types 274
measure_with_unit
AIM EXPRESS expanded listing entities 315
AIM EXPRESS short listing imported entity modifications 125
mechanical part
definition 8
mechanical_context
AIM EXPRESS expanded listing entities 315
AIM EXPRESS short listing entities 105
minute_in_hour
AIM EXPRESS expanded listing types 274
mixed_loop_type_set
AIM EXPRESS expanded listing functions 381
model_name
application object attribute 28
mapping table 68
month_in_year_number
AIM EXPRESS expanded listing types 274
msb_shells
AIM EXPRESS expanded listing functions 381
msf_curve_check
AIM EXPRESS expanded listing functions 381
msf_surface_check
AIM EXPRESS expanded listing functions 383
named_unit
AIM EXPRESS expanded listing entities 315

AIM EXPRESS short listing imported entity modifications 128
next_assembly_usage_occurrence
AIM EXPRESS expanded listing entities 315
AIM EXPRESS short listing imported entity modifications 130
mapping table 48
no_shape_for_make_from
AIM EXPRESS expanded listing rules 346
AIM EXPRESS short listing rules 191
no_shape_for_supplied_part
AIM EXPRESS expanded listing rules 346
AIM EXPRESS short listing rules 192
non_topological_surface_and_wireframe
application object 22
mapping table 71
unit of functionality 13
normalise
AIM EXPRESS expanded listing functions 384
offset_curve_3d
AIM EXPRESS expanded listing entities 315
offset_surface
AIM EXPRESS expanded listing entities 316
open_shell
AIM EXPRESS expanded listing entities 316
open_shell_reversed
AIM EXPRESS expanded listing functions 385
ordered_action
mapping_table 55
ordinal_date
AIM EXPRESS expanded listing entities 316
organization
AIM EXPRESS expanded listing entities 316
application object attribute 25
mapping table 44, 85
organization_relationship
AIM EXPRESS expanded listing entities 316
organizational_address
AIM EXPRESS expanded listing entities 316
organizational_project
AIM EXPRESS expanded listing entities 316
oriented_closed_shell
AIM EXPRESS expanded listing entities 317
oriented_edge

AIM EXPRESS expanded listing entities 317
oriented_face
AIM EXPRESS expanded listing entities 317
oriented_open_shell
AIM EXPRESS expanded listing entities 318
oriented_path
AIM EXPRESS expanded listing entities 318
orthogonal_complement
AIM EXPRESS expanded listing functions 385
outer_boundary_curve
AIM EXPRESS expanded listing entities 318
parabola
AIM EXPRESS expanded listing entities 318
parameter_value
AIM EXPRESS expanded listing types 275
parametric_representation_context
AIM EXPRESS expanded listing entities 319
AIM EXPRESS short listing imported entity modifications 129
part
application assertion 35, 36
application object 22
mapping table 67, 74-76, 79
part_classification
application object attribute 22
mapping table 74
part_identification
unit of functionality 14
part_nomenclature
application object attribute 23
mapping table 74
part_number
application object attribute 23
mapping table 74
part_type
application object attribute 23
mapping table 74
part_version
application assertion 35, 36, 38
application object 23
mapping table 46, 57, 59, 75, 76, 78
path
AIM EXPRESS expanded listing entities 319

path_head_to_tail
AIM EXPRESS expanded listing functions 385
path_reversed
AIM EXPRESS expanded listing functions 386
pcurve
AIM EXPRESS expanded listing entities 319
pcurve_or_surface
AIM EXPRESS expanded listing types 275
person
AIM EXPRESS expanded listing entities 319
AIM EXPRESS short listing imported entity modifications 124
application object attribute 25
mapping table 44
person_and_organization
AIM EXPRESS expanded listing entities 319
person_and_organization_assignment
AIM EXPRESS expanded listing entities 320
person_and_organization_role
AIM EXPRESS expanded listing entities 320
AIM EXPRESS short listing imported entity modifications 124
person_organization
application assertion 33, 36, 38
application object 24
mapping table 43, 45, 46, 60
person_organization_item
AIM EXPRESS expanded listing types 275
AIM EXPRESS short listing types 103
mapping table 45, 46, 60, 86
person_organization_select
AIM EXPRESS expanded listing types 275
personal_address
AIM EXPRESS expanded listing entities 320
phase_of_product
application object attribute 28
mapping table 67
placement
AIM EXPRESS expanded listing entities 320
plane
AIM EXPRESS expanded listing entities 320
plane_angle_measure
AIM EXPRESS expanded listing types 275
plane_angle_measure_with_unit

AIM EXPRESS expanded listing entities 320
plane_angle_unit
AIM EXPRESS expanded listing entities 320
planned_date_effectivity
application object 25
mapping table 64
planned_effectivity
application assertion 34, 36
application object 26
mapping table 48, 64, 67
planned_lot_effectivity
application object 26
mapping table 64
planned_sequence_effectivity
application object 26
mapping table 65
point
AIM EXPRESS expanded listing entities 321
point_on_curve
AIM EXPRESS expanded listing entities 321
point_on_surface
AIM EXPRESS expanded listing entities 321
point_replica
AIM EXPRESS expanded listing entities 321
poly_loop
AIM EXPRESS expanded listing entities 321
polyline
AIM EXPRESS expanded listing entities 321
positive_length_measure
AIM EXPRESS expanded listing types 275
positive_plane_angle_measure
AIM EXPRESS expanded listing types 275
preferred_surface_curve_representation
AIM EXPRESS expanded listing types 275
process_specification
application object 27
mapping table 61
product
AIM EXPRESS expanded listing entities 322
AIM EXPRESS short listing imported entity modifications 118
mapping table 74, 83, 84
product_category

AIM EXPRESS expanded listing entities 322
AIM EXPRESS short listing imported entity modifications 117
mapping table 74, 75
product_category_relationship
AIM EXPRESS expanded listing entities 322
product_concept
AIM EXPRESS expanded listing entities 322
AIM EXPRESS short listing imported entity modifications 129
mapping table 67, 68
product_concept_context
AIM EXPRESS expanded listing entities 322
product_concept_requires_configuration_item
AIM EXPRESS expanded listing rules 346
AIM EXPRESS short listing rules 181
product_configuration
application assertion 36
application object 27
mapping table 67, 68
product_context
AIM EXPRESS expanded listing entities 322
AIM EXPRESS short listing imported entity modifications 116
product_definition
AIM EXPRESS expanded listing entities 323
AIM EXPRESS short listing imported entity modifications 118
mapping table 45, 72
product_definition_context
AIM EXPRESS expanded listing entities 323
AIM EXPRESS short listing imported entity modifications 117
mapping table 73
product_definition_effectivity
AIM EXPRESS expanded listing entities 323
product_definition_formation
AIM EXPRESS expanded listing entities 323
AIM EXPRESS short listing imported entity modifications 118
product_definition_formation_with_specified_source
AIM EXPRESS expanded listing entities 323
product_definition_relationship
AIM EXPRESS expanded listing entities 323
mapping table 47, 73, 76
product_definition_requires_approval
AIM EXPRESS expanded listing rules 346
AIM EXPRESS short listing rules 152

product_definition_requires_date_time
AIM EXPRESS expanded listing rules 346
AIM EXPRESS short listing rules 153
product_definition_requires_person_organization
AIM EXPRESS expanded listing rules 347
AIM EXPRESS short listing rules 151
product_definition_shape
AIM EXPRESS expanded listing entities 323
product_definition_usage
AIM EXPRESS expanded listing entities 324
AIM EXPRESS short listing imported entity modifications 128
product_definition_with_associated_documents
AIM EXPRESS expanded listing entities 324
product_model
application assertion 36
application object 28
mapping table 67, 68
product_related_product_category
AIM EXPRESS expanded listing entities 324
AIM EXPRESS short listing imported entity modifications 117
mapping table 74
product_requires_person_organization
AIM EXPRESS expanded listing rules 347
AIM EXPRESS short listing rules 147
product_requires_product_category
AIM EXPRESS expanded listing rules 347
AIM EXPRESS short listing rules 135
product_requires_version
AIM EXPRESS expanded listing rules 347
AIM EXPRESS short listing rules 147
product_version
mapping table 75-78
mapping_table 46
product_version_requires_approval
AIM EXPRESS expanded listing rules 347
AIM EXPRESS short listing rules 148
product_version_requires_person_organization
AIM EXPRESS expanded listing rules 348
AIM EXPRESS short listing rules 149
product_version_requires_security_classification
AIM EXPRESS expanded listing rules 348
AIM EXPRESS short listing rules 150

promissory_usage_occurrence
AIM EXPRESS expanded listing entities 324
mapping table 51
property_definition
AIM EXPRESS expanded listing entities 324
property_definition_representation
AIM EXPRESS expanded listing entities 324
purpose
application object attribute 16
mapping table 42
quantified_assembly_component_usage
AIM EXPRESS expanded listing entities 325
quantity_unit_of_measure
mapping table 66
quasi_uniform_curve
AIM EXPRESS expanded listing entities 325
quasi_uniform_surface
AIM EXPRESS expanded listing entities 325
rational_b_spline_curve
AIM EXPRESS expanded listing entities 325
rational_b_spline_surface
AIM EXPRESS expanded listing entities 325
reason
application object attribute 33
mapping table 57
recommended_solution
application object attribute 17
mapping table 54
rectangular_composite_surface
AIM EXPRESS expanded listing entities 325
rectangular_trimmed_surface
AIM EXPRESS expanded listing entities 326
reference_designator
application object attribute 21
mapping table 49
release_status
application object attribute 24
mapping table 77
reparametrised_composite_curve_segment
AIM EXPRESS expanded listing entities 326
representation
AIM EXPRESS expanded listing entities 326

AIM EXPRESS short listing imported entity modifications 126
representation_context
AIM EXPRESS expanded listing entities 327
AIM EXPRESS short listing imported entity modifications 127
representation_item
AIM EXPRESS expanded listing entities 327
AIM EXPRESS short listing imported entity modifications 128
representation_map
AIM EXPRESS expanded listing entities 327
representation_relationship
AIM EXPRESS expanded listing entities 327
representation_relationship_with_transformation
AIM EXPRESS expanded listing entities 327
request_date
application object attribute 33
mapping table 58
requested_action
mapping_table 54, 57, 58
restrict_action_request_status
AIM EXPRESS expanded listing rules 348
AIM EXPRESS short listing rules 144
restrict_approval_status
AIM EXPRESS expanded listing rules 348
AIM EXPRESS short listing rules 158
restrict_certification_type
AIM EXPRESS expanded listing rules 348
AIM EXPRESS short listing rules 154
restrict_contract_type
AIM EXPRESS expanded listing rules 349
AIM EXPRESS short listing rules 160
restrict_date_time_role
AIM EXPRESS expanded listing rules 349
AIM EXPRESS short listing rules 167
restrict_document_type
AIM EXPRESS expanded listing rules 349
AIM EXPRESS short listing rules 169
restrict_person_organization_role
AIM EXPRESS expanded listing rules 349
AIM EXPRESS short listing rules 166
restrict_product_category_value
AIM EXPRESS expanded listing rules 349
AIM EXPRESS short listing rules 133

restrict_security_classification_level
AIM EXPRESS expanded listing rules 350
AIM EXPRESS short listing rules 165
reversible_topology
AIM EXPRESS expanded listing types 276
reversible_topology_item
AIM EXPRESS expanded listing types 276
revision_letter
application object attribute 24
mapping table 77
scalar_times_vector
AIM EXPRESS expanded listing functions 386
schema identification 415
seam_curve
AIM EXPRESS expanded listing entities 327
second_in_minute
AIM EXPRESS expanded listing types 276
second_proj_axis
AIM EXPRESS expanded listing functions 387
security_classification
AIM EXPRESS expanded listing entities 328
AIM EXPRESS short listing imported entity modifications 123
security_classification_assignment
AIM EXPRESS expanded listing entities 328
security_classification_level
AIM EXPRESS expanded listing entities 328
AIM EXPRESS short listing imported entity modifications 123
mapping table 48, 78
security_classification_optional_date_time
AIM EXPRESS expanded listing rules 350
AIM EXPRESS short listing rules 164
security_classification_requires_approval
AIM EXPRESS expanded listing rules 350
AIM EXPRESS short listing rules 161
security_classification_requires_date_time
AIM EXPRESS expanded listing rules 350
AIM EXPRESS short listing rules 163
security_classification_requires_person_organization
AIM EXPRESS expanded listing rules 350
AIM EXPRESS short listing rules 162
security_code
application object attribute 19, 24

mapping table 48, 78
serial_numbered_effectivity
AIM EXPRESS expanded listing entities 328
mapping table 65, 66
set_of_reversible_topology_item
AIM EXPRESS expanded listing types 276
set_of_topology_reversed
AIM EXPRESS expanded listing functions 387
shape
application assertion 37
application object 28
mapping table 80, 81
unit of functionality 14
shape_aspect
AIM EXPRESS expanded listing entities 328
application assertion 37
application object 28
mapping table 81, 82
shape_aspect_relationship
AIM EXPRESS expanded listing entities 328
shape_definition
AIM EXPRESS expanded listing types 276
shape_definition_ representation
mapping_table 80
shape_definition_representation
AIM EXPRESS expanded listing entities 329
mapping table 82
shape_representation
AIM EXPRESS expanded listing entities 329
AIM EXPRESS short listing imported entity modifications 127
mapping table 80
shape_representation_relationship
AIM EXPRESS expanded listing entities 329
shell
AIM EXPRESS expanded listing types 276
shell_based_surface_model
AIM EXPRESS expanded listing entities 329
shell_based_wireframe_model
AIM EXPRESS expanded listing entities 329
shell_based_wireframe_shape_representation
AIM EXPRESS expanded listing entities 329
shell_reversed

AIM EXPRESS expanded listing functions 387
si_prefix
AIM EXPRESS expanded listing types 277
si_unit
AIM EXPRESS expanded listing entities 332
si_unit_name
AIM EXPRESS expanded listing types 277
solid model
definition 8
solid_angle_measure
AIM EXPRESS expanded listing types 278
solid_angle_measure_with_unit
AIM EXPRESS expanded listing entities 333
solid_angle_unit
AIM EXPRESS expanded listing entities 333
solid_model
AIM EXPRESS expanded listing entities 333
source
AIM EXPRESS expanded listing types 278
source_control
unit of functionality 14
specification
application assertion 33, 37
application object 29
mapping table 61, 62, 82
specification_code
application object attribute 29
mapping table 62
specification_source
application object attribute 29
mapping table 62
specification_usage_constraint
mapping table 62
specified_higher_usage_occurrence
AIM EXPRESS expanded listing entities 333
specified_item
AIM EXPRESS expanded listing types 278
AIM EXPRESS short listing types 104
mapping table 73, 82
spherical_surface
AIM EXPRESS expanded listing entities 334
standard_part_indicator

application object attribute 23
mapping table 75
start_date
mapping table 64
start_order
application object 29
mapping table 55
start_request
AIM EXPRESS expanded listing entities 334
AIM EXPRESS short listing entities 108
application object 29
mapping table 55
start_request_item
AIM EXPRESS expanded listing types 278
AIM EXPRESS short listing types 101
mapping table 59
start_request_requires_approval
AIM EXPRESS expanded listing rules 351
AIM EXPRESS short listing rules 140
start_request_requires_date_time
AIM EXPRESS expanded listing rules 351
AIM EXPRESS short listing rules 141
start_request_requires_person_organization
AIM EXPRESS expanded listing rules 351
AIM EXPRESS short listing rules 140
start_work
AIM EXPRESS expanded listing entities 334
AIM EXPRESS short listing entities 109
mapping table 55
start_work_requires_approval
AIM EXPRESS expanded listing rules 351
AIM EXPRESS short listing rules 142
start_work_requires_date_time
AIM EXPRESS expanded listing rules 351
AIM EXPRESS short listing rules 143
status
application object attribute 16, 33
mapping table 42, 58
sub-assembly
definition 8
substitute_part
application assertion 34, 35

application object 29
mapping table 48, 51, 76
subtype_mandatory_action
AIM EXPRESS expanded listing rules 352
AIM EXPRESS short listing rules 172
subtype_mandatory_effectivity
AIM EXPRESS expanded listing rules 352
AIM EXPRESS short listing rules 182
subtype_mandatory_product_context
AIM EXPRESS expanded listing rules 352
AIM EXPRESS short listing rules 131
subtype_mandatory_product_definition_formation
AIM EXPRESS expanded listing rules 352
AIM EXPRESS short listing rules 173
subtype_mandatory_product_definition_usage
AIM EXPRESS expanded listing rules 352
AIM EXPRESS short listing rules 185
subtype_mandatory_representation
AIM EXPRESS expanded listing rules 353
AIM EXPRESS short listing rules 190
subtype_mandatory_representation_context
AIM EXPRESS expanded listing rules 353
AIM EXPRESS short listing rules 190
subtype_mandatory_shape_representation
AIM EXPRESS expanded listing rules 353
AIM EXPRESS short listing rules 188
supplied_part
application assertion 35, 37
application object 30
mapping table 79, 83, 85, 86
supplied_part_relationship
AIM EXPRESS expanded listing entities 334
AIM EXPRESS short listing entities 107
mapping table 79
supplier
application assertion 36, 37
application object 31
mapping table 46, 85, 86
supplier_id
mapping table 85
supplier_part_number
application object attribute 30

mapping table 84
supported_item
AIM EXPRESS expanded listing types 278
surface
AIM EXPRESS expanded listing entities 334
surface_curve
AIM EXPRESS expanded listing entities 334
surface_finish_specification
application object 31
mapping table 62
surface_model
AIM EXPRESS expanded listing types 278
surface_of_linear_extrusion
AIM EXPRESS expanded listing entities 335
surface_of_revolution
AIM EXPRESS expanded listing entities 335
surface_patch
AIM EXPRESS expanded listing entities 335
surface_replica
AIM EXPRESS expanded listing entities 335
surface_weights_positive
AIM EXPRESS expanded listing functions 388
swept_surface
AIM EXPRESS expanded listing entities 335
text
AIM EXPRESS expanded listing types 278
thru_effectivity_id
application object attribute 27
mapping table 66
topological_representation_item
AIM EXPRESS expanded listing entities 336
topology_reversed
AIM EXPRESS expanded listing functions 388
toroidal_surface
AIM EXPRESS expanded listing entities 336
transformation
AIM EXPRESS expanded listing types 278
application object attribute 18
mapping table 47
transition_code
AIM EXPRESS expanded listing types 279
trimmed_curve

AIM EXPRESS expanded listing entities 336
trimming_preference
AIM EXPRESS expanded listing types 279
trimming_select
AIM EXPRESS expanded listing types 279
uncertainty_measure_with_unit
AIM EXPRESS expanded listing entities 336
uniform_curve
AIM EXPRESS expanded listing entities 336
uniform_surface
AIM EXPRESS expanded listing entities 337
unique_version_change_order
AIM EXPRESS expanded listing functions 389
AIM EXPRESS short listing functions 194
unique_version_change_order_rule
AIM EXPRESS expanded listing rules 354
AIM EXPRESS short listing rules 146
unit
AIM EXPRESS expanded listing types 279
unit_of_measure
application object attribute 21
mapping table 49
usage_constraint
application assertion 37
application object 31
mapping table 62
using_items
AIM EXPRESS expanded listing functions 389
using_representations
AIM EXPRESS expanded listing functions 390
valid_calendar_date
AIM EXPRESS expanded listing functions 391
valid_geometrically_bounded_wf_curve
AIM EXPRESS expanded listing functions 391
valid_geometrically_bounded_wf_point
AIM EXPRESS expanded listing functions 392
valid_measure_value
AIM EXPRESS expanded listing functions 392
valid_time
AIM EXPRESS expanded listing functions 393
valid_units
AIM EXPRESS expanded listing functions 393

valid_wireframe_edge_curve
AIM EXPRESS expanded listing functions 395
valid_wireframe_vertex_point
AIM EXPRESS expanded listing functions 395
value
application object attribute 31
mapping table 63
vector
AIM EXPRESS expanded listing entities 337
vector_difference
AIM EXPRESS expanded listing functions 396
vector_or_direction
AIM EXPRESS expanded listing types 279
version
application object attribute 17
mapping table 54
versioned_action_request
AIM EXPRESS expanded listing entities 337
AIM EXPRESS short listing imported entity modifications 119
versioned_action_request_requires_solution
AIM EXPRESS expanded listing rules 354
AIM EXPRESS short listing rules 145
versioned_action_request_requires_status
AIM EXPRESS expanded listing rules 354
AIM EXPRESS short listing rules 145
vertex
AIM EXPRESS expanded listing entities 337
vertex_loop
AIM EXPRESS expanded listing entities 337
vertex_point
AIM EXPRESS expanded listing entities 337
vertex_shell
AIM EXPRESS expanded listing entities 337
volume_measure
AIM EXPRESS expanded listing types 279
volume_measure_with_unit
AIM EXPRESS expanded listing entities 337
volume_unit
AIM EXPRESS expanded listing entities 338
week_in_year_number
AIM EXPRESS expanded listing types 279
week_of_year_and_day_date

AIM EXPRESS expanded listing entities 338
wire_shell
AIM EXPRESS expanded listing entities 338
wireframe model
definition 8
wireframe_model
AIM EXPRESS expanded listing types 279
wireframe_with_topology
application object 31
mapping table 87
unit of functionality 15
work_item
AIM EXPRESS expanded listing types 280
AIM EXPRESS short listing types 101
work_order
application assertion 38
application object 32
mapping table 55, 57
work_order_id
application object attribute 32
mapping table 55
work_request
application assertion 38
application object 32
mapping table 57, 59, 60
work_request_id
application object attribute 33
mapping table 58
year_number
AIM EXPRESS expanded listing types 280