Recommended Practices for Model Styling and Organization

Release 1.1

September 5, 2011

Contacts

Jochen Boy	Phil Rosché
ProSTEP iViP Association	PDES, Inc.
Dolivostraße 11	5300 International Blvd.
64293 Darmstadt / Germany	North Charleston, SC 29418 USA
jochen.boy@prostep.com	phil.rosche.ctr@ati.org

Table of Contents

List of Figures

List of Tables

Document History

This document replaces the following CAx-IF Recommended Practices:

- Recommended Practices for Colors and Layers;
  published September 24, 2001
- Recommended Practices for Assembly Instance Styling; Release 1.0;
  published November 19, 2002
- Recommended Practices for Colors, Layers and Groups; Draft Release 1.1;
  published December 1, 2008

The current document covers the scope of the preceding ones, and adds new and updated concepts.

1 Introduction

This document describes the recommended practices for implementing the ability to exchange information about model styling – colors and visibility – and model organization – layers and groups – via the STEP standard.

The support of Colors and Layers is standard functionality in most STEP processors for many years. These capabilities support a better presentation of the model on the screen, and an organization of the elements in the model as per the user companies' guidelines. Groups represent additional organizational structures within the model supported by some CAD systems.

This document also describes the recommended practices for implementing the ability to assign context dependent styles to individual instances of a component in an assembly via the STEP standard.

The approaches described hereafter have been combined from previously separate, but related, documents and updated to the latest agreements and changes in the data structure.

2 Scope

The following are within the scope of this document:

- The specification of colors for solids and topologically bounded surfaces as well as their constituent shape elements, and also geometrically bounded wireframe and surface data
- The assignment of styles (colors and visibility) to instances of a component in an assembly
- The definition of layers and groups

The following are out of scope for this document:

- The definition of "Saved Views" (as defined in digital product definition standards) for model viewing organization. These are described in the "Recommended Practices for the Representation and Presentation of Product and Manufacturing Information (PMI)".
- The definition of styling information for "high end" visualization, such as surface texture and transparency.

3 Model Styling

The following sections cover the aspects of displaying the model described in the STEP file in the intended way. This is typically done by assigning colors at the top level, which are then inherited down the model structure to the individual geometric elements. For individual elements, this styling can be overridden in general, or in a specific context. The applicable styles also include invisibility.

3.1 Global Styling Container

It is a general convention in STEP – to be precise in Part46 – that only styled items shall be displayed when viewing the contents of a STEP file. That means that all geometric elements that do not have any style assigned at all shall be invisible.

If a style is assigned, there are two basic options: either, an explicit style is assigned – for instance a specific color – defining the way the model shall be shown, or a "null" style is used, meaning that it is up to the receiving system to determine the way the data is displayed. As this concept is quite important, it is summarized in the following table:

Assigned Style	Model or model element is displayed….
none	not at all
"null" style	according to target system preferences (i.e. default colors)
explicit style (color,…)	as defined in the STEP file

Table 1: Styling choices

In every STEP file, there shall be one global styling container, meaning an entity that displays all information about the initial display of the model. This container can be one of two entities:

- `draughting_model` ("DM")
- `mechanical_design_geometric_presentation_representation` ("MDGPR")

As there may be more than one DM or MDGPR in a STEP file, it is important to clearly identify the one acting as the global styling container. This is achieved in the following way:

- The `name` attribute is an empty string ('')
- If there are other instances of DM or MDGPR in the file, the global one is always referenced by the `rep_2` attribute of `mechanical_design_and_draughting_-relationship` (or its supertype, `representation_relationship`).
- Any occurrences of `draughting_model_item_association` will refer to the global DM.

Note that any additional items in the set of items of the DM or MDGPR besides the part geometry (e.g. annotations, camera models,…) as well as any advanced implementation approaches including `presentation_set` and `presentation_area` are out of scope for this document. Please refer to the "Recommended Practices for the Representation and Presentation of Product and Manufacturing Information (PMI)", section 10.4 ("Saved Views") for further details.

Figure 1 on the next page illustrates the minimum set of display information required in a STEP file. It defines that the entire part shape shall be displayed, by linking the top-level `shape_-representation` (typically `advanced_brep_shape_representation` ("ABSR") for solid models) to the global styling container. Definition of the `null_style` means that the target system's preferred or default settings will be applied to display the data.

The structure in Figure 1 shall be created for all top-level representations containing geometry. This also applies to supplemental geometry (`constructive_geometry_representation`, cp. corresponding Recommended Practices), as well as surface or wireframe models. For an assembly, it is sufficient to include the top node `shape_representation`. By convention this will then apply to all child nodes as well.

Important Note The styling container and the shape representations mapped into it need to share the same `geometric_representation_context`. The `mapped_item.mapping_-target` and `representation_map.mapping_origin` therefore shall define a unit transformation, i.e. with (0/0/0) as origin and the unit vectors as directions.

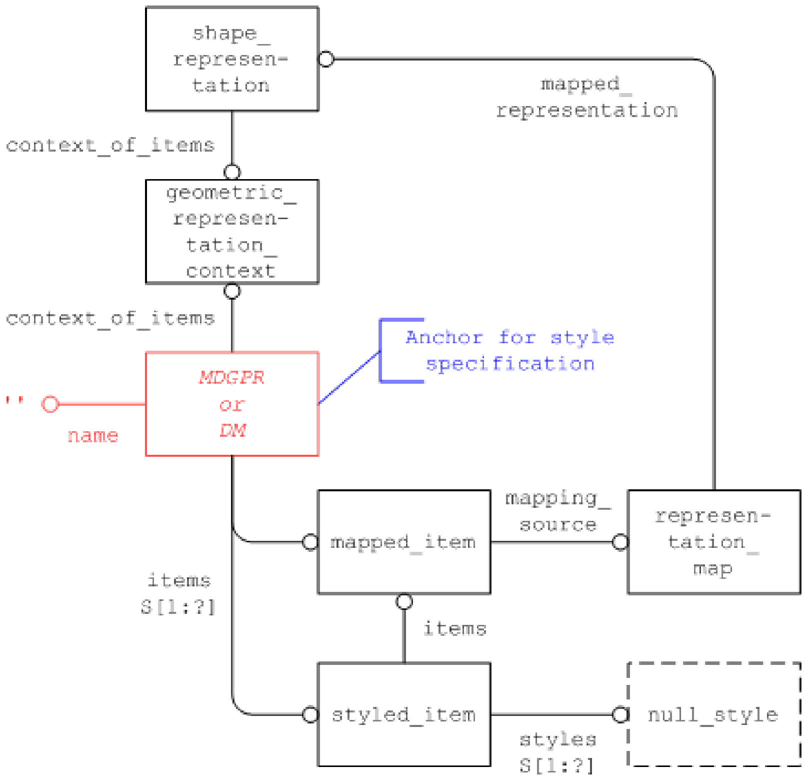

Figure 1: Global Styling Container Definition

3.2 Coloring of Solids, Surfaces and Curves

3.2.1 Priority and inheritance of model colors

The coloring information is always inherited from the solid or surface to its constituent faces and edges. The priority list for coloring elements in topological models is as follows:

1. solids (`manifold_solid_brep`, `brep_with_voids`) OR surfaces (`shell_based_surface_model`)
2. surfaces only (`open_shell`, `closed_shell`)
3. faces / edges
4. geometric surfaces / curves

Solids and surfaces may be colored by using `fill_area_style_colour`, where faces lying on a solid should be styled by overriding the solid style, as are the edges. The edges are treated as being independent of the face due to them potentially being used by two faces. This could lead to a conflict when the two faces were colored differently. See sections 3.2.4 and 3.4 for handling of overriding styles.

In order to maintain similarity of style, it was decided at the 7th CAx Implementor Forum meeting to extend this approach to include geometrically bounded surfaces and wireframes. Thus, the priority list for coloring these elements is as follows:

1. Wireframe / Surface collector (`geometric_set`, `geometric_curve_set`)
2. Geometric Representation Item (e.g. `trimmed_curve`, `b_spline_surface`)

Colors should be instantiated from the top down for these hierarchies, e.g. the solid should always be colored, then any differences in face colors applied by overriding the solid color for the different face. Similarly for wireframe, the `geometric_set` / `geometric_curve_set` should always be colored in the majority color for the geometric entities, those deviating from this majority color being overridden.

3.2.2 Color Instantiation

Figure 2 below illustrates the structure to assign a specific color to a model element:

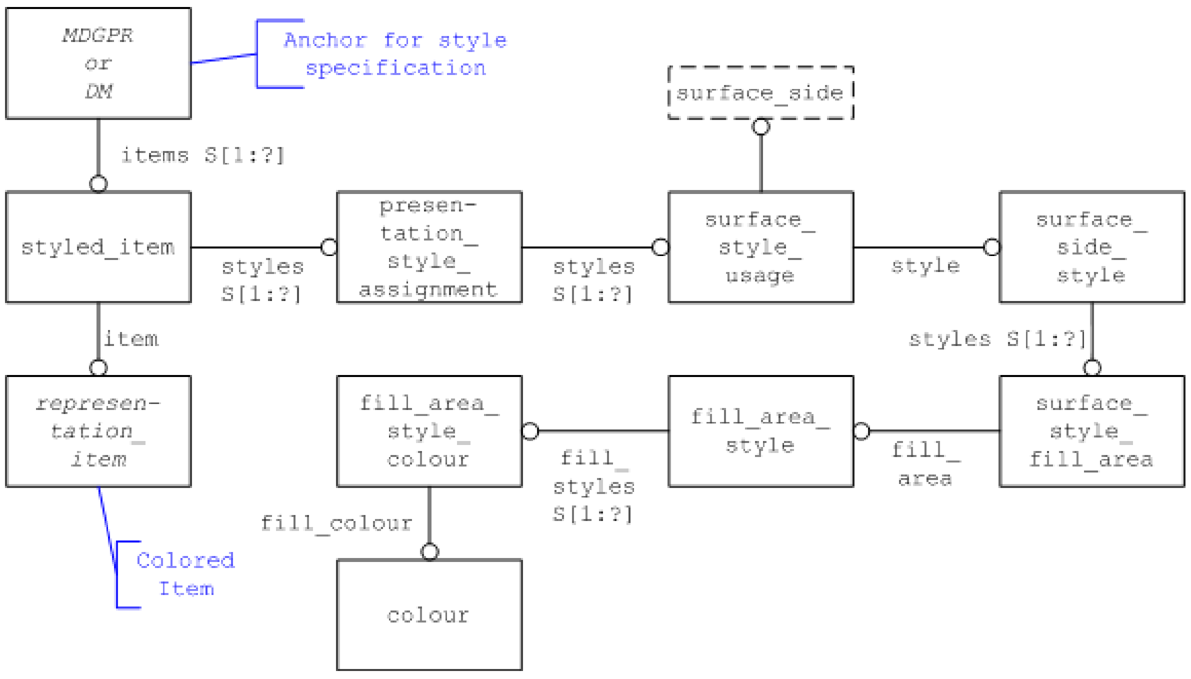

Figure 2: Color Representation for Surfaces / Solids

Part21 Example:

```
#164=MANIFOLD_SOLID_BREP('',#163);
#226=DRAUGHTING_PRE_DEFINED_COLOUR('cyan');
#227=FILL_AREA_STYLE_COLOUR('',#226);
#228=FILL_AREA_STYLE('',(#227));
#229=SURFACE_STYLE_FILL_AREA(#228);
#230=SURFACE_SIDE_STYLE('',(#229));
#231=SURFACE_STYLE_USAGE(.BOTH.,#230);
#232=PRESENTATION_STYLE_ASSIGNMENT((#231));
#233=STYLED_ITEM('',(#232),#164);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268,#273,),#269);
```

3.2.3 Pre-Defined Colors

Table 2 below lists the pre-defined colors along with the RGB values that shall be assumed for them:

Color	RGB
Black	0, 0, 0
White	1, 1, 1
Red	1, 0, 0
Green	0, 1, 0
Blue	0, 0, 1
Yellow	1, 1, 0
Cyan	0, 1, 1
Magenta	1, 0, 1

Table 2: RGB Values for Pre-Defined Colors

3.2.4 Invisibility

Invisibility is a capability that will allow hiding of model elements. It is basically handled in the same way as any simple style (e.g. colors). Under the boundary conditions given in section 3.1 above, the usage convention is quite simple: Unless invisibility is present, all elements are visible. The structure for this is very simple: an instance of invisibility will be linked to the `styled_item` shown on the left hand side of Figure 2:

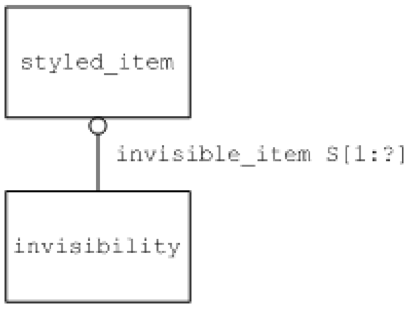

Figure 3: Structure of the Invisibility Assignment

The `styled_item` points to a `presentation_style_by_context` (to transport the information that the invisibility is assigned in the context given) which needs to point to a certain style. In order to make sure the appearance of the component is not affected in systems not supporting invisibility, the `presentation_style_by_context` shall reference the style originally assigned to the component.

3.3 Overriding Surface Colors

For coloring of a surface lying on a solid, the color of the solid is overridden. This happens by using an instance of `over_riding_styled_item`. Only portions of shape redefined by `over_riding_styled_item` will be affected. Portions not redefined by the overriding style shall be handled as already defined

Part21 Example:

```
#150=ADVANCED_FACE('',(#144),#149,.T.);
#247=DRAUGHTING_PRE_DEFINED_COLOUR('magenta');
#248=FILL_AREA_STYLE_COLOUR('',#247);
#249=FILL_AREA_STYLE('',(#248));
#250=SURFACE_STYLE_FILL_AREA(#249);
#251=SURFACE_SIDE_STYLE('',(#250));
#252=SURFACE_STYLE_USAGE(.BOTH.,#251);
#253=PRESENTATION_STYLE_ASSIGNMENT((#252));
#254=OVER_RIDING_STYLED_ITEM('',(#253),#150,#233);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268,#273),#269);
```

Figure 4 illustrates the entity structure to define and overriding surface color.

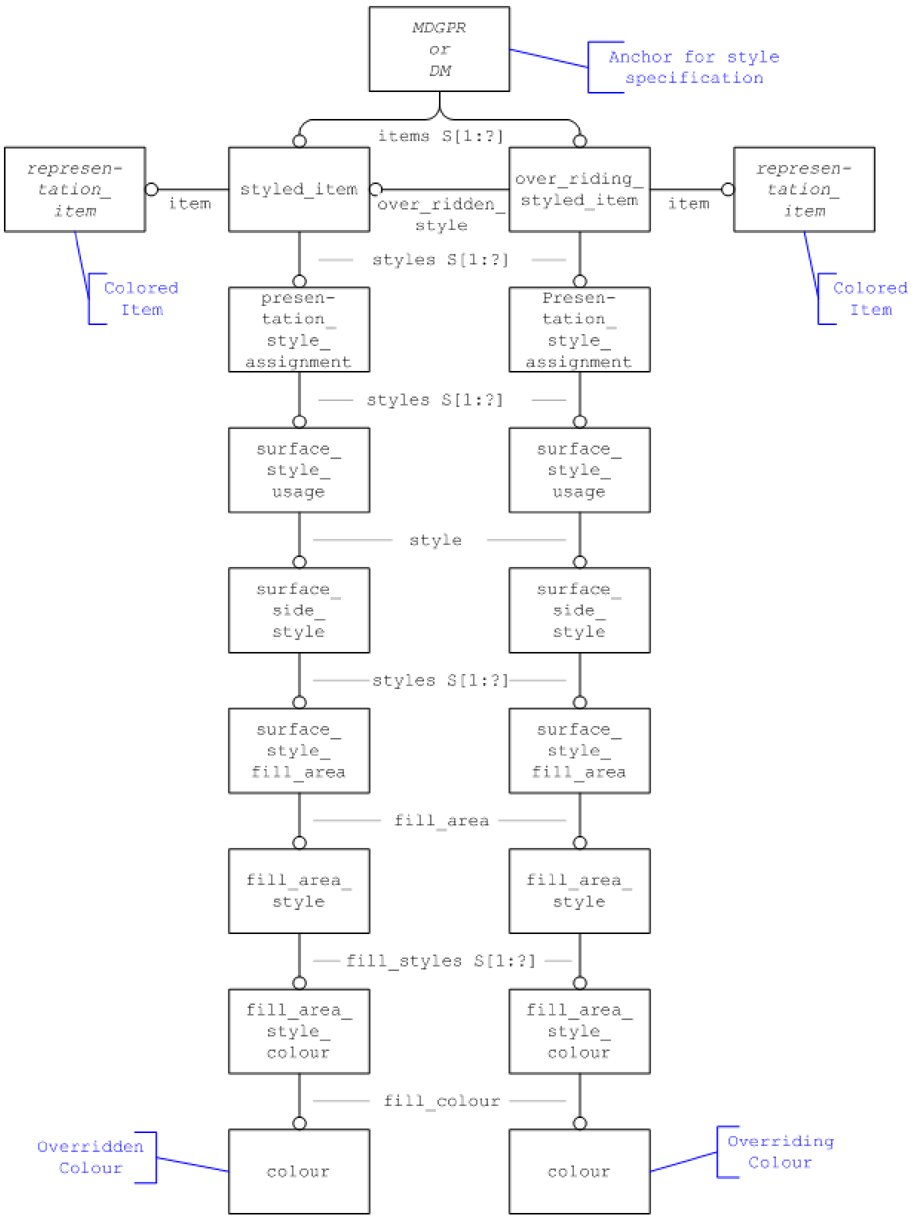

Figure 4: Representation of Overriding Surface Color

3.4 Overriding Edge Colors

For coloring of an edge, the color of the solid or surface is overridden. This happens by using an instance of `over_riding_styled_item`.

Note that edge colors are applied by a `curve_style` rather than a `surface_style`. This leads to a situation where the `surface_style` of the solid or surface containing the edge could be overridden by a `curve_style` for the edge in question.

Only portions of shape redefined by `over_riding_styled_item` will be affected. Portions not redefined by the overriding style shall be handled as already defined.

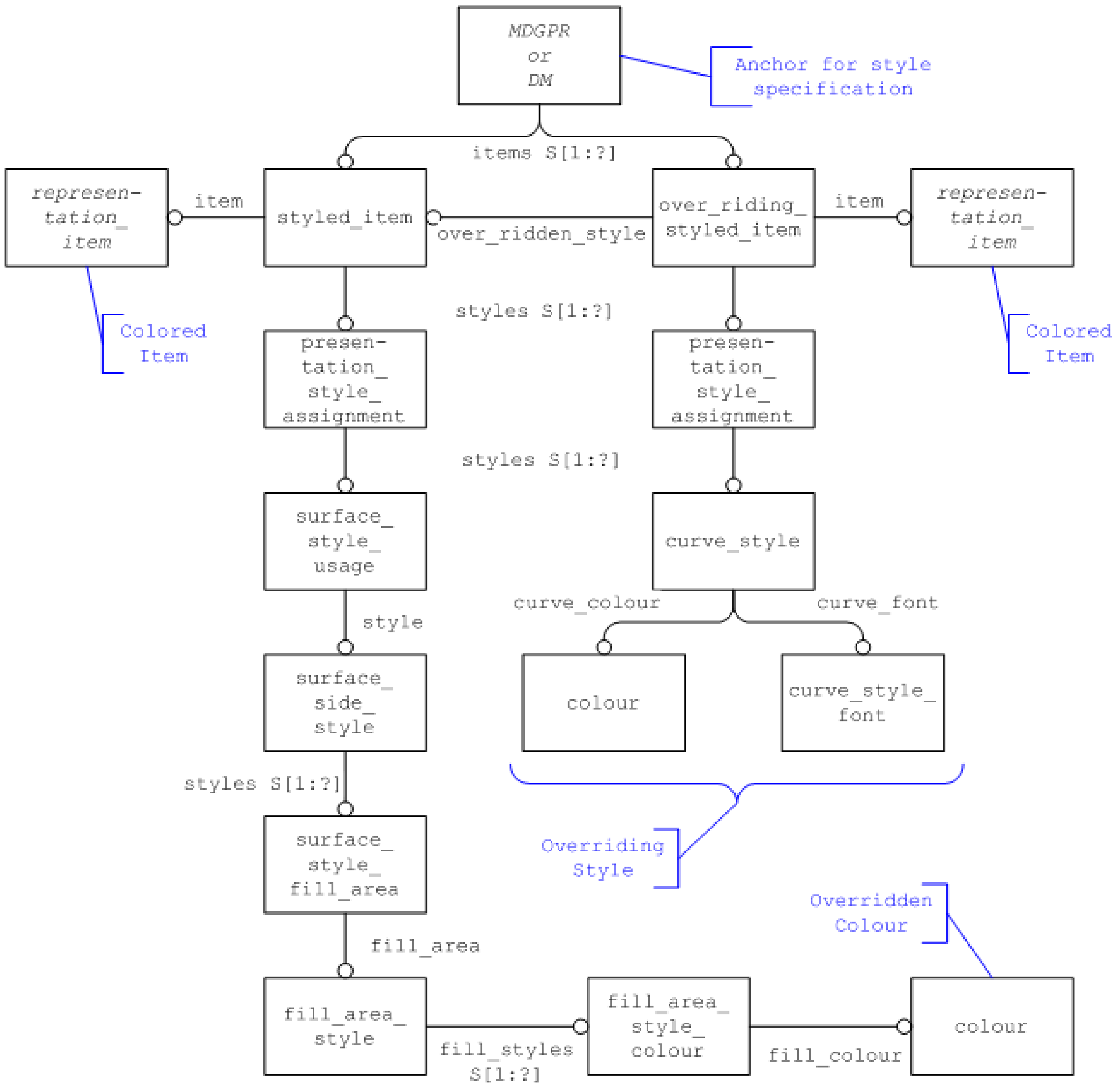

Figure 5: Representation of Overriding Edge Color

Part21 Example:

```
#87=EDGE_CURVE('',#28,#24,#66,.T.);
#242=DRAUGHTING_PRE_DEFINED_COLOUR('yellow');
#243=DRAUGHTING_PRE_DEFINED_CURVE_FONT('continuous');
#244=CURVE_STYLE('',#243,POSITIVE_LENGTH_MEASURE(1.0),#242);
#245=PRESENTATION_STYLE_ASSIGNMENT((#244));
#246=OVER_RIDING_STYLED_ITEM('',(#245),#87,#233);
#276=DRAUGHTING_MODEL('#276',(#233,#241,#246,#254,#255,#263,#268,#273),#269);
```

4 Assembly Instance Styling

The scope is the assignment of new styles to certain components of an assembly, in order to emphasize certain parts in a given context. The style assigned is either a new color or an “invisibility” tag, which will declare the respective component as hidden.

4.1 Linking the Style to the Component Instance

The main item of interest in the context of assembly instance styling is the identification of the correct instance. There are two cases which have to be considered and are illustrated by the following picture, which relates to the well-known AS1 model:

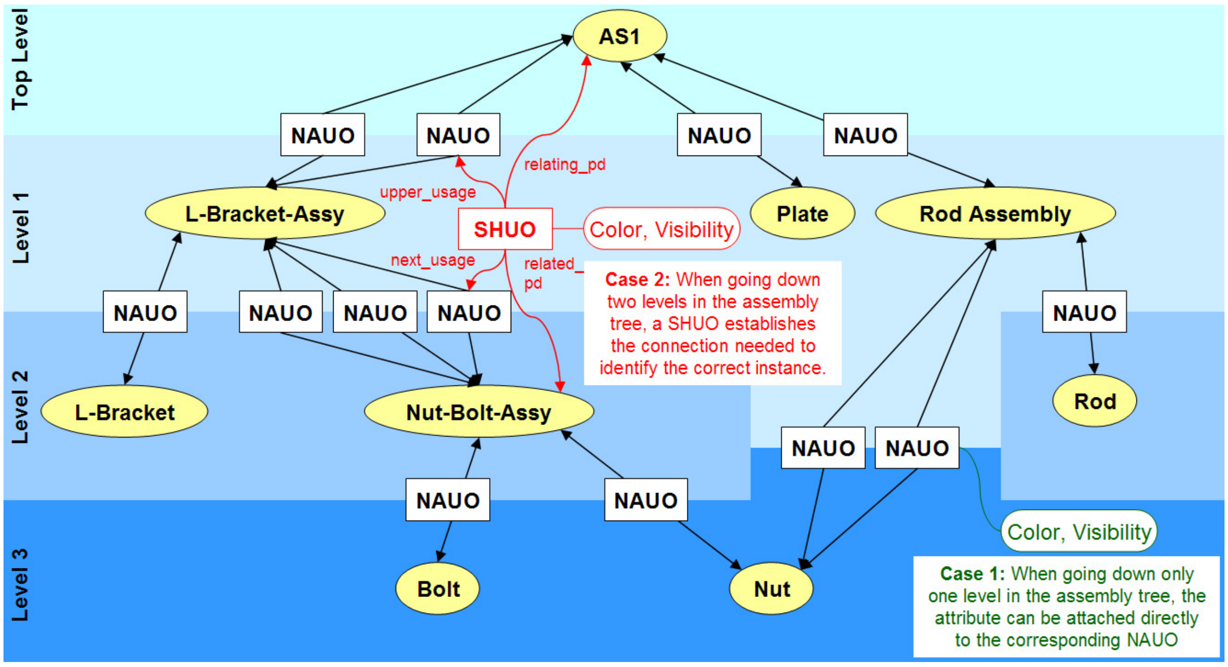
Top Level
Level 1
Level 2
Level 3
AS1
NAUO
L-Bracket-Assy
Plate
Rod Assembly
SHUO
Color, Visibility
relating_pd
upper_usage
next_usage
related_pd
Case 2: When going down two levels in the assembly tree, a SHUO establishes the connection needed to identify the correct instance.
L-Bracket
Nut-Bolt-Assy
Rod
Bolt
Nut
Case 1: When going down only one level in the assembly tree, the attribute can be attached directly to the corresponding NAUO

Figure 6: Identification of the Instance to be Styled

The two key entities used here are:

- NAUO = `next_assembly_usage_occurrence`
- SHUO = `specified_higher_usage_occurrence`

Note that there is a limitation here in that the usage of SHUO only works if the entire assembly structure is given in one file. There is no way yet to identify component instances deep down in the assembly tree in combination with “nested” external references.

4.2 Simple Case (NAUO Approach)

The simple case of identifying the instance is if the instance is directly referenced by the assembly which creates the context. In Figure 6 above this can be seen in the bottom right corner, where the Rod Assembly references the Nut.

In this case, the style information is attached to the `next_assembly_usage_occurrence` (NAUO) which establishes the relationship. The diagram and file excerpt below show how this is done in STEP:

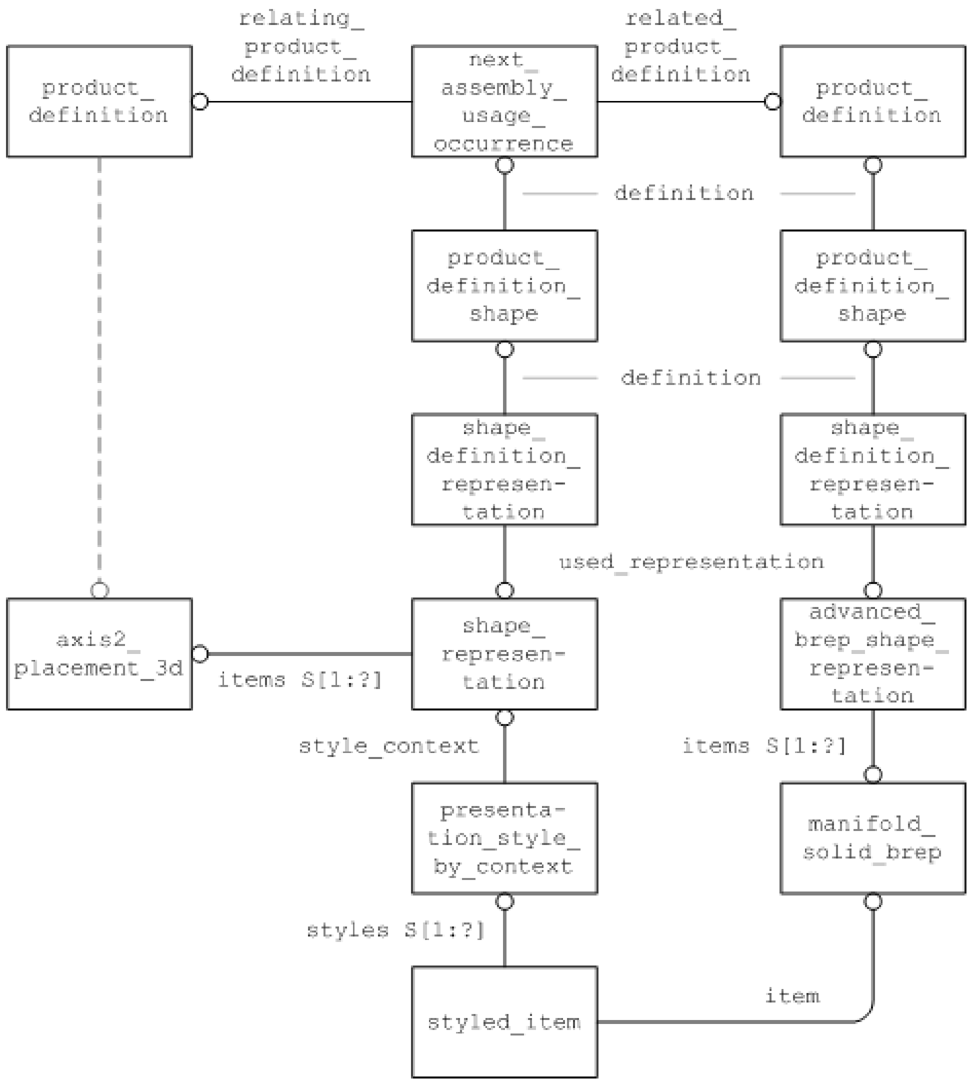
relating_ product_ definition
related_ product_ definition
product_ definition
next_ assembly_ usage_ occurrence
product_ definition
definition
product_ definition_ shape
product_ definition_ shape
definition
shape_ definition_ represen- tation
shape_ definition_ represen- tation
used_representation
axis2_ placement_3d
items S[1:?]
shape_ represen- tation
advanced_ brep_shape_ represen- tation
style_context
items S[1:?]
presenta- tion_style_ by_context
manifold_ solid_brep
styles S[1:?]
styled_item
item

Figure 7: Structure of the Assignment of a Style to the NAUO

The link between the NAUO and the style is established by the usual chain of `product_definition_shape`, `shape_definition_representation` and `shape_representation`. The style is then defined through a `presentation_style_by_context`, as opposed to its supertype `presentation_style_assignment` used in the “original” style for the shape. Finally, a `styled_item` links the style to its shape.

The applied style itself is defined through the `presentation_style_by_context`, see section 4.4 below.

Part21 Example:

```
#13=PRODUCT_DEFINITION('Produkt1',' ',#6,#3);
#25=SHAPE_REPRESENTATION(' ',(#200,#215,#228,#241,#254,#30),#208);
#26=SHAPE_DEFINITION_REPRESENTATION(#193,#25);
#29=PRODUCT_DEFINITION('Part1',' ',#28,#3);
#30=MANIFOLD_SOLID_BREP('Manifold Brep',#40);
#193=PRODUCT_DEFINITION_SHAPE(' ',' ',#29);
#209=NEXT_ASSEMBLY_USAGE_OCCURRENCE('Part1.2','Part1.2',' ',#13,#29,' ');
#210=PRODUCT_DEFINITION_SHAPE(' ',' ',#209);
#214=AXIS2_PLACEMENT_3D(' ',#217,#221,#220);
#262=SHAPE_DEFINITION_REPRESENTATION(#210,#263);
#263=SHAPE_REPRESENTATION(' ',(#214),#208);
#264=PRESENTATION_STYLE_BY_CONTEXT((#266),#263);
#265=STYLED_ITEM(' ',(#264),#30);
```

4.3 Extended Case (SHUO Approach)

If there is more than one level in the assembly tree between the instance to be styled and the assembly definition giving the context, it is not possible to clearly identify the desired instance by simply picking one NAUO.

In this case, `specificed_higher_usage_occurrence` (SHUO) entities are needed. A single SHUO will allow identifying an instance two levels down in the assembly structure. If it is necessary to go down even further, a chain of SHUOs needs to be created bottom-up, i.e. the higher SHUOs reference the lower ones. In the example above, if a specific instance of a Nut shall be identified from the root node, the structure would look like this:

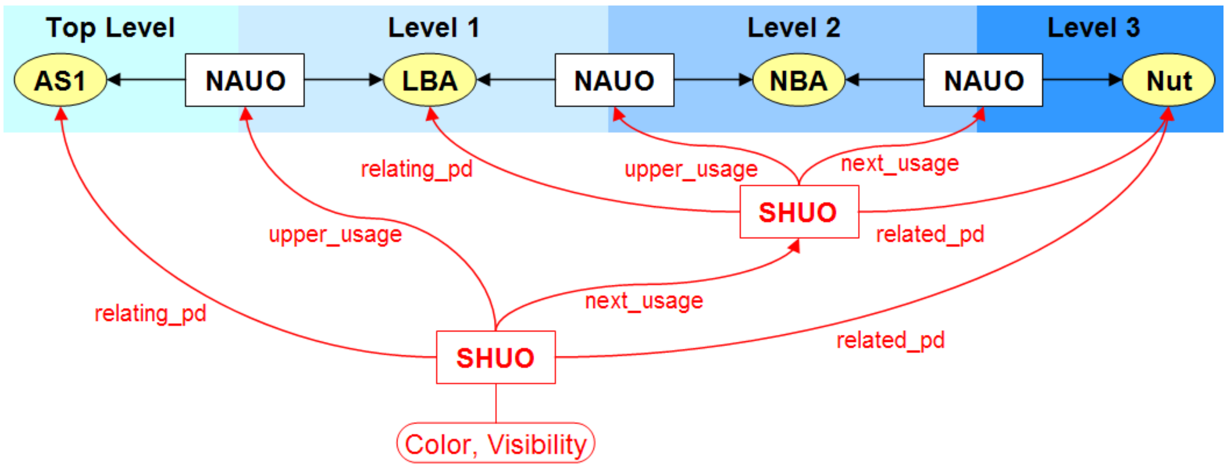

Figure 8: Finding the Path through the Assembly using SHUO

<u>Note</u> that each SHUO may be created only once, since there is a uniqueness rule saying that the combination of `upper_usage` and `next_usage` shall be unique within the file. Each SHUO can be used, however, to associate a number of user defined attributes or other properties (such as styles) to the component instance.

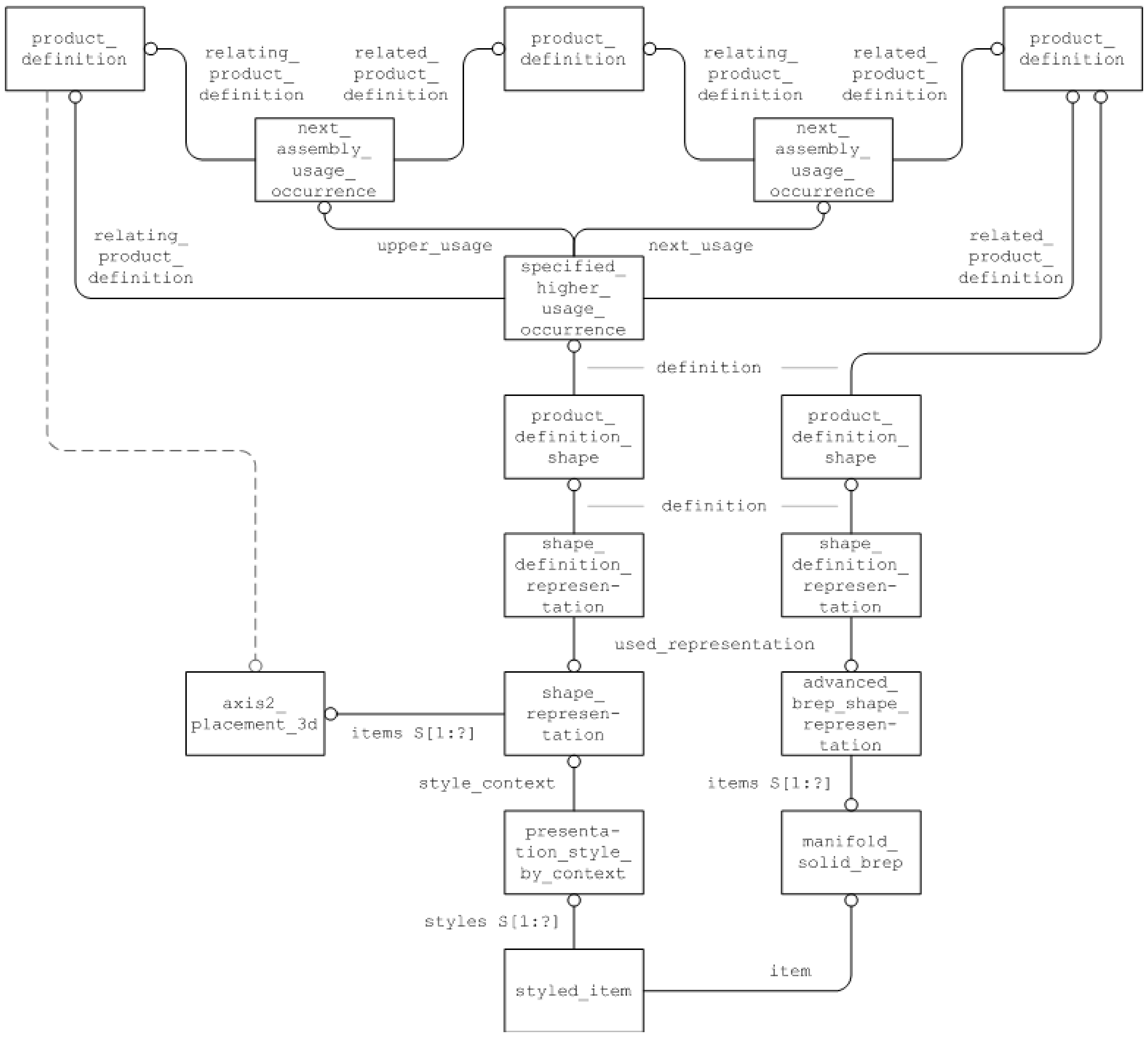
product_definition
relating_product_definition
related_product_definition
product_definition
relating_product_definition
related_product_definition
product_definition
next_assembly_usage_occurrence
next_assembly_usage_occurrence
upper_usage
next_usage
relating_product_definition
specified_higher_usage_occurrence
related_product_definition
definition
product_definition_shape
product_definition_shape
definition
shape_definition_represen-tation
shape_definition_represen-tation
used_representation
axis2_placement_3d
items S[1:?]
shape_represen-tation
advanced_brep_shape_represen-tation
style_context
items S[1:?]
presenta-tion_style_by_context
manifold_solid_brep
styles S[1:?]
styled_item
item

Figure 9: Structure of the Assignment of a Style via a SHUO

4.4 Applicable Styles

The two styling characteristics described in this document that can be used to distinguish one instance of a component from its other instances are color and visibility.

4.4.1 Color

The assignment of a new color happens in the same way already described in section 3.2 above. The `presentation_style_by_context` in Figure 7 or Figure 9 references the style which finally defines the new color for the component in the given context, cp. Figure 2 starting at `presentation_style_assignment`.

4.4.2 Invisibility

For invisibility, the definitions given in 3.2.4 apply here as well. In the context of assembly instance styling, it can be used to hide a specific instance of a component. The structure for this is again very simple: an instance of invisibility will be linked to the `styled_item` shown at the bottom of Figure 7 and Figure 9.

5 Layers

A layer is a general structure for the collection of geometric and annotation elements. Layers shall not be nested, i.e.; a layer cannot be put on another layer.

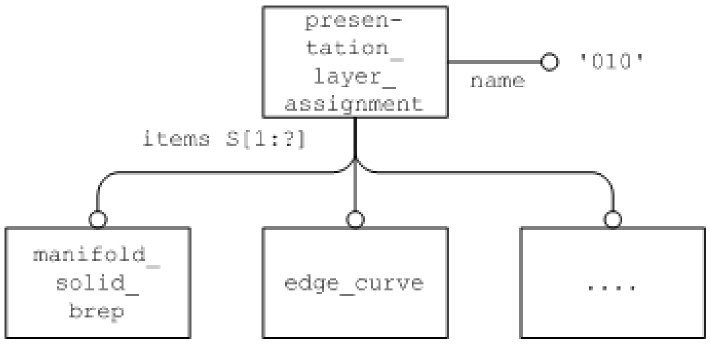

Figure 10: Layer Representation

The `presentation_layer_assignment` entity collects all items that are on the same layer in its assigned `items` attribute. These items are `representation_items`.

Note that this ignores the informal proposition on `layered_item` provided in Part46. This approach was chosen by the Implementor Forum in order to reduce exchange file size by removing the need for `layered_items` to be `styled_items`, although a STEP file which applies layers through `styled_items` is not deemed to be invalid.

In addition, a whole layer can be set to invisible, using an instance of `invisibility` referencing the `presentation_layer_assignment`. **Note** that even though all unstyled items are considered to be invisible, in the case of intended invisibility the invisible objects shall be declared explicitly as invisible. This may happen directly for each object or indirectly via declaring the layer that contains the objects as invisible.

Part21 Example:

```
#225=PRESENTATION_LAYER_ASSIGNMENT('010','layer 010',(#206));
#206=SHELL_BASED_SURFACE_MODEL('#206',(#205));
```

6 Groups

A group is an organizational structure that allows the grouping of specific elements in the model together. Relationships can be defined between groups to create a group hierarchy.

6.1 Assigning Elements to a Group

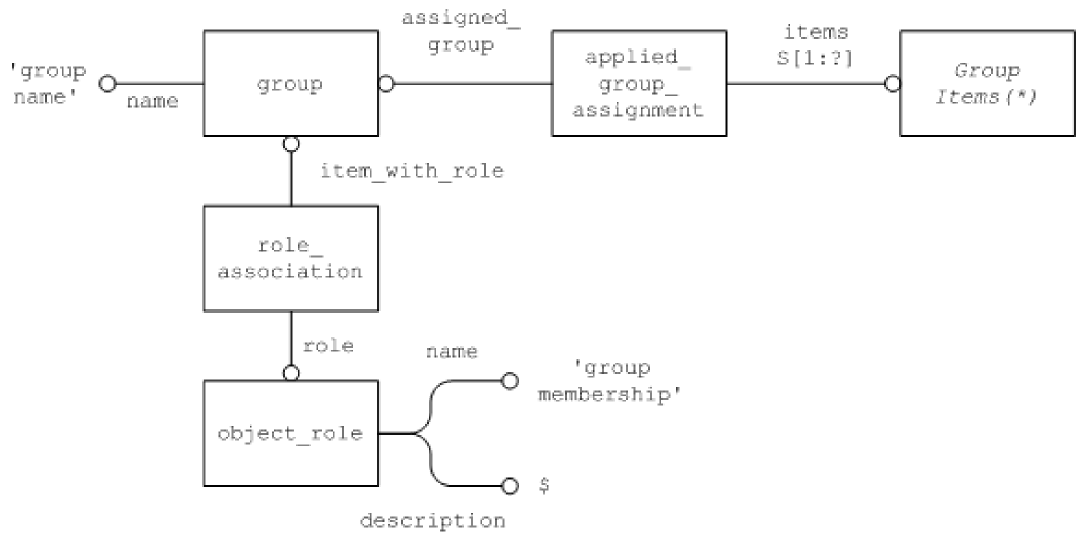

Figure 11: Group Assignment Representation

(*) **Note:** Even though `applied_group_assignment` is contained in both AP203e2 and AP214, the select types for `group_item` (AP214) and `groupable_item` (AP203e2) are different. AP203e2 allows more entity types to be grouped together. In AP214, a where rule further limits the groupable items to `geometric_representation_items` and `shape_aspects`.

As long as no further business requirements are known, the items in a group shall be limited to the AP214 definition.

6.2 Defining Group Relationships

Groups may be used to define other groups, and thus to create a group hierarchy.

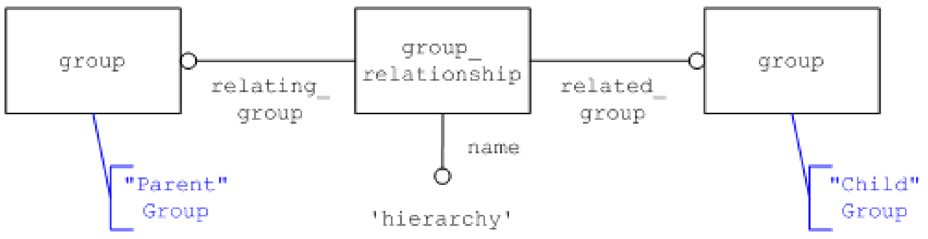

Figure 12: Group Relationship Representation

Annex A Part 21 File Examples

STEP files relating to the capabilities described in this document are available in the public STEP File Library on the CAx-IF homepage; see either http://www.cax-if.de/library/ or http://www.cax-if.org/library/.

The files are based on current schemas for both AP203 Edition 2 and AP214, and have been checked for syntax and compliance with the Recommended Practices.

Annex B Availability of implementation schemas

B.1 AP214

The AP214 schemas support the implementation of the capabilities as described. The schemas can be retrieved from:

- IS Version (2001) – http://www.cax-if.de/documents/ap214_is_schema.zip
- 3rd Edition (2010) – http://www.cax-if.de/documents/AP214E3_2010.zip

B.2 AP203 2nd Edition

The long form EXPRESS schema for the second edition of AP203 can be retrieved from:

- http://www.cax-if.de/documents/part403ts_wg3n2635mim_lf.exp

Note that the first edition of AP203 is no longer support in the Recommended Practices.

B.3 AP242

The capabilities described in this document are also supported by AP242, the upcoming joint successor of AP203 and AP214, with no changes to the instantiation.

The latest development longform EXPRESS schema for AP242 can be found in the CAx-IF member area. It will be published on the public web site once approved by ISO.