Methodology for Harmonisation of Modules

by

Andries van Renssen

andries.s.h.vanrenssen@opc.shell.com

Version 0.1

27 January 2001

(For review at ISO meeting In Funchal)

Reference: WG10 N331

Table of Content

1. Introduction

This document discusses a proposal for a methodology for STEP Modules Harmonisation. The proposal uses a draft Intelligent Schematics module(s) as an example.

2. Specialisation of modules (subtyping)

A complete harmonised set of modules should ideally consist of one integrated specialisation/generalisation hierarchy of concepts. This implies that every entity in a module should be an explicit subtype of another entity in that module, or an explicit subtype of an external entity (an entity external to the module).

Harmonisation between modules therefore includes that concepts used in one module should be subtypes of concepts used in another module.
From this it becomes clear that a set of top level modules with top level entities (concepts) is required. The modules dependency diagram of figure 1 illustrates this requirement.

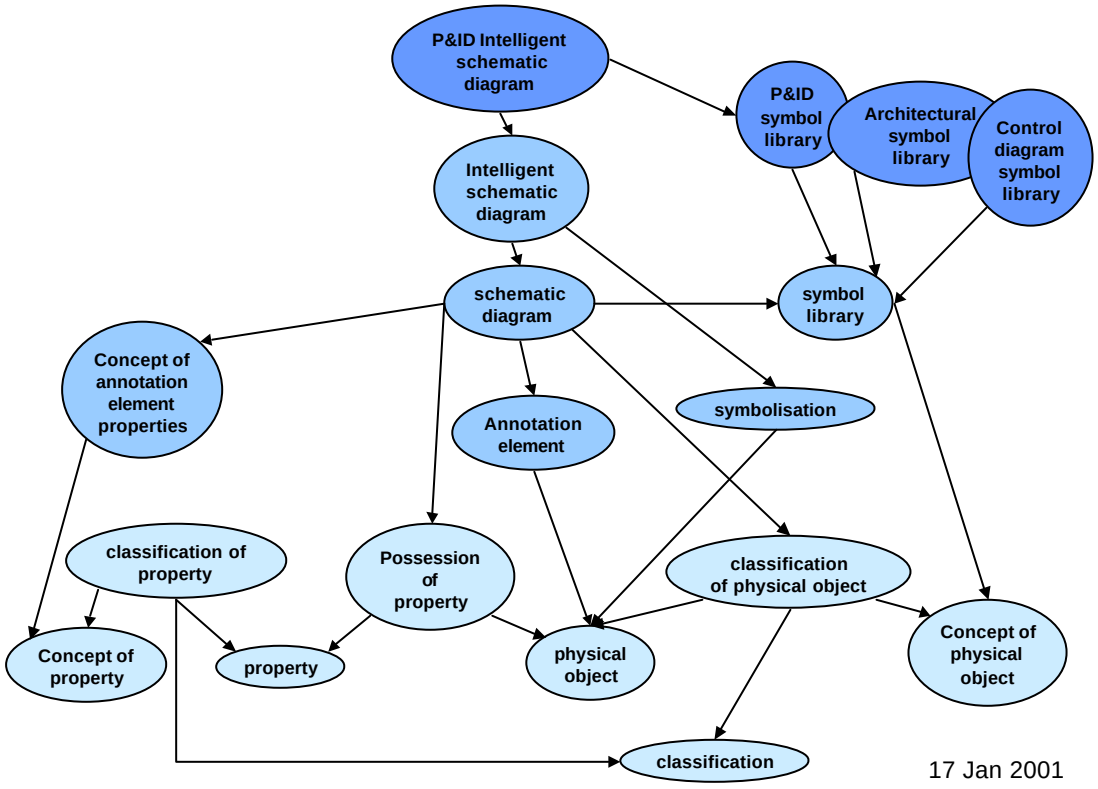

Figure 1, Dependency between modules

Figure 1 shows that an intelligent schematics module depends on the existence of schematics, which depends on various other concepts, such as classification. So, a module might reveal the need to define missing higher level modules in order to complete the hierarchy.

This is illustrated by the draft part of an Intelligent Schematics module (see figure 2).

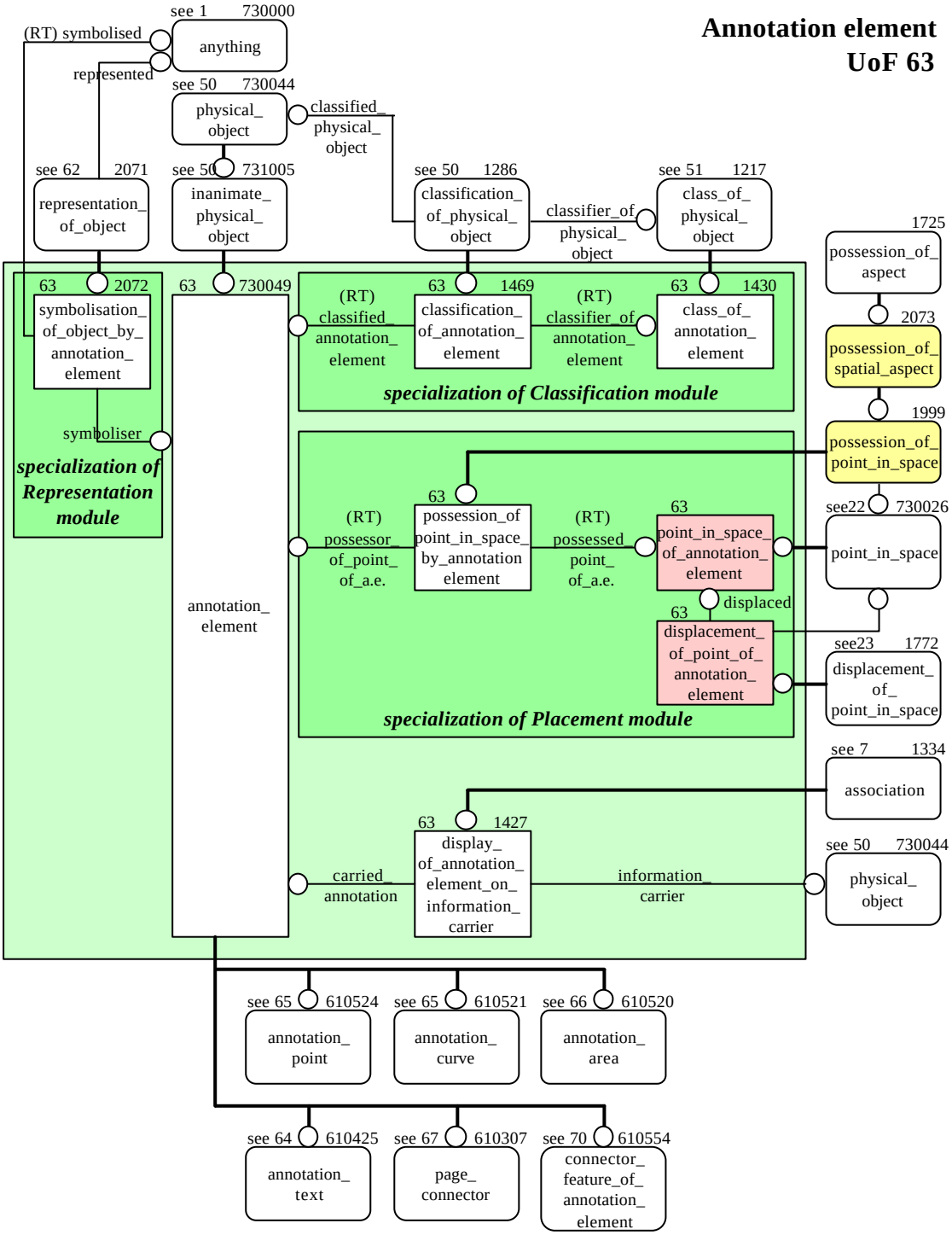

Figure 2, Annotation element harmonised module

In figure 2, the module that needs to be harmonised is put in the context of its environment. The module itself consists of the part within the light green box. The entities outside that box are ‘external entities’ for the module. For example, individual annotation elements (the 'ink on paper') can be classified according to the type of things they symbolise. Examples of classes of symbols are: line, box, (shaded) area, text (annotation text), pump symbol, etc. This classification is defined by the entity classification_of_annotation_element, which appears to be a subtype of the entity classification_of_physical_object (which ideally should exist already in another module).

Similarly, the entity class_of_annotation_element is a subtype of class_of_physical_object. This subtype defines a subset of classes of physical objects that may validly by used to classify annotation elements. To define such a subset it usually is sufficient to define only the top of a sub-hierarchy, but sometimes the classes are constrained to a "picklist" of classes. In the latter case the picklist should be defined as an explicit constraint on the instances of the supertype class.

So, harmonisation of the module shows the requirement that other modules should exist for the higher level concepts from which the concepts in the module are subtypes.

This suggests that it seems appropriate to start from atomic modules that contain one concept only and build molecular modules from that.

An example of an existing hierarchy of concepts is called STEPlib. It contains a specialisation hierarchy with many concepts, including most concepts from which the Intelligent Schematics module concepts are subtypes. It is not just a traditional class library, but it includes also higher level concept entities in the hierarchy. As such is can be viewed as a data model concept hierarchy. It is proposed as a starter set of a concept hierarchy for modules. It is available on /www.steplib.com/.

3. Inheritance and attributes

The strict subtyping as described above brings a special issue with it which result from the fact that subtype entities inherit the attributes of their supertype entities. Often this inheritance is an advantage, but often it is also a disadvantage, especially when the choice of attributes that the supertype has differs from the choice of attributes that are wanted for the subtype. Furthermore, one module may define something as an attribute of an entity where another proposed module may define a separate entity (with its own attributes) for the same concept.

A traditional route for harmonisation in such situations is to try to harmonise the choice of attributes. However, this may be possible for some existing models, but is impossible with respect to future still unknown requirements.

A way out to this problem is to harmonise in the following four steps:

1. To harmonise only on (attribute-less) concepts. This means that concepts are structured in a specialisation hierarchy. They don't inherit attributes. So, no issues will arise because of unwanted inheritance. But instead of inheritance of attributes the concepts inherit the capabilities (options!) that the associations that are defined for the supertype concept are instantiated.
2. To harmonise the attributes by treating them as separate concepts and arranging them in a specialisation hierarchy as well.
3. To harmonise the relationships between the entities and the attributes by treating them as separate concepts and to harmonise them also in a specialisation hierarchy.
4. To define templates that group concepts into attributed entities.

This is illustrated in the Intelligent Schematics module part about annotation_text.
Assume we started with an attributed entity, let me call it annotation_text_template. This

entity has three attributes:
- textual_encoded_information,
- text_appearance and
- text_box.

The steps above then result in the following:

1. The concept of the entity is called annotation_text and is defined as a subtype of annotation_element.
2. The three attributes are harmonised by defining them as subtypes of encoded_information, property and box_2d, whereas a set (library) of valid text appearances is defined and where it is discovered that text_box is in fact a role of a box_2d. The latter means that the attribute text_box could better be renamed as box_2d.
3. The three relationships are harmonised by defining them subtypes of other possession concepts as indicated in the diagram.
4. The diagram of figure 3 is constructed that defines the attributed entity as a group of harmonised concepts.

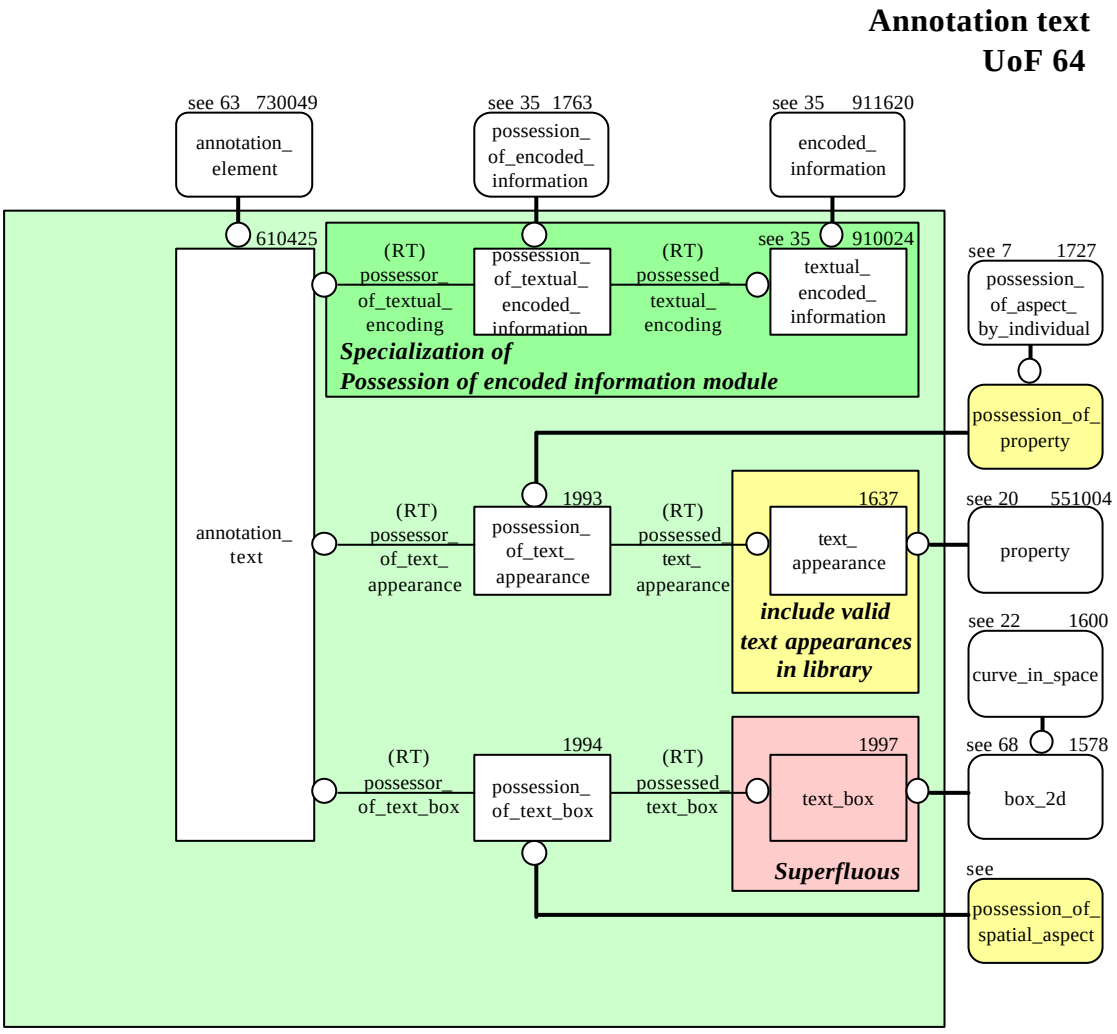

Figure 3, Annotation text - attributed entity

N.B. The conceptual modelling style (as developed in EPISTLE) can be of assistance here, because it defines entities that represent concepts (without attributes), whereas the idea of an attribute is replaced by a combination of an association entity and another entity that represents the attribute. In that style no attributes are inherited, but instead the entities inherit the (optional!) capability that the associations that are defined for the supertype concept are instantiated.

The separate associations provide the possibility that such associations can be instantiated more than once and can have additional information such as duration of validity. The multiple instantiation is limited by cardinality constraints.

An attributed entity in the traditional modelling style can be transformed in a template of concepts in a conceptual modelling style.

Once a concept entity is defined as a subtype of a higher level concept it should be verified what 'capabilities' the subtype does inherit from its supertype. In the example of the Intelligent Schematics module the annotation_element is defined to be a sub-subtype of physical_object, which means that an annotation element inherits the capabilities that a physical object have. For example, assume that the capability to form compositions of physical objects is defined in a supertype module that includes an entity called composition_of_physical_object which relates two physical objects with the roles part and whole.

Now the module developer has two options:

1. He includes a subtype of composition_of_physical_object in the module.
2. He does not include that subtype in the module.

For an AP developer the difference will be that in the first option inclusion of the intelligent schematics module in the AP implies that composition of annotation elements is included automatically. In the second case the decision whether to include the "composition of physical object" capability is left to an AP developer, because an AP developer can choose whether he wants to include the supertype module in his AP or not.

From this example it can be concluded that if a capability is an important aspects of a module, then it is recommended to include a subtype of the more generic capability concept in the module.

4. Interface entities

The concept of a reusable module includes that an application protocol (AP) that needs the functionality that is already defined in an existing module does not need to define that requirement again, but it could reuse the module and "integrate" it in the AP.

Similarly harmonisation and integration of modules includes that a module that needs the functionality that is already defined in an existing module should be interfaced. Integration of a new module with an existing module should be done by the definition of one or more **interface entities** that relate entities in the new module to entities of the existing module. Such an interface entity can be present already within the existing the module, where it defines a relationship with an **external entity** (an entity that is external to the module). It is also possible that the new module defines a completely new interface entity between one of its own entities and an entity in the existing module.

When a new module considers to reuse an interface entity that exists already in the existing module then there are three options:

1. The new module contains an entity that is identical to the external entity of the existing module.
2. The new module contains an entity that is a subtype of the external entity and therefore the new module also has to define a subtype of the interface entity that exists in the module.
3. The new module contains an entity that is a supertype of the external entity and thus the new module intends to generalise the use of the existing module and should define the interface entity of the existing module as a subtype of the generalised interface entity in the new module.
   If the generalisation of the interface entity appears to be generally valid, then it should be considered to include the generalisation in the existing module.

The same process applies if an AP wants to use an existing module. Then the new AP should be interfaced to the module as described above.
For example, assume that a new proposed module (or an AP) wants to reuse the Intelligent Schematics module(s) to schematically symbolise building constructions. Therefore the entity "building_component" was included in the module as the object to be symbolised. The module can then be integrated with the intelligent schematics module by the definition of an interface entity that relates the building_component to the entity annotation_element from the Intelligent Schematics module (see figure 2).

However, the intelligent schematics module already includes the concept of symbolisation of object by annotation element. Therefore the interface entity in the new module appears to be a subtype of the entity in the module called symbolisation_of_object_by_annotation_element.

Furthermore, the entity building_component appears to be a subtype of the entity application_object that was used as an external entity by the module.

Now, the module (or AP) developer has two options:

1. He can include the supertype entity application_object also in his new module and thus generalises the objects that can be symbolised (which might be useful anyway in order to symbolise also properties such as e.g. a slope).
2. He defines his entity to be a subtype of the external entity (which then remains also external to his new module) and define his own interface entity and make it an explicit subtype of the interface entity of the module.

This illustrated the following general requirement for modules:
Modules should preferably include interface entities that define relationships with external entities (external to the module) and modules should refer to those external entities explicitly and indicate where they are defined.

This is illustrated in the intelligent schematic module by the interface entity called display_of_annotation_element_on_information_carrier. This entity relates an annotation element to the external entity physical_object.
A module (or AP) developer who includes the intelligent schematics module should

therefore either include the entity physical_object in his module (or AP), or he should include a subtype of it. For example he could include "paper" when he wants to constrain the information carriers to paper only. In the latter case he should also define a subtype of the display entity in his module (or AP).

So, module harmonisation is a kind of integration (or interfacing) between modules that is similar to integration of a module in an AP.

PS I forgot to mention in my note that proposed instances for "class_of_annotation_element" can be found as a specialisation hierarchy under the concept 'annotation element' in the spreadsheet anno02.xls on www.steplib.com.