Title:
Criticism of the STEP Application Module approach

Version 0.1, draft
Document identification: ISO TC184/SC4 WG12 N3776
Release date: 2005-06-05
Author: Lothar Klein, LKSoftWare GmbH

1. Introduction

In the SC4 standing document "SC4 Industrial Data Framework", ISO TC184/SC4 N1167, 2001-08-01 we can read:

> *"The adoption of Application Modules (AMs) allows a set of core concepts to be defined once and interpreted once. ... Other AMs can then make use of these concepts without the need for the concept to be re-interpreted. Application Protocols developed using these AMs will then be interoperable, provided they conform to certain guidelines or rules."*

We must see that this goal is only partially reached by the modules we have today. The main reason is that the required "*certain guidelines or rules*" are missing and as a consequence too many AM are not interoperable between different APs. The goal of this paper is to identify some areas where further improvement and corrections in the current set of modules is essential to achieve better interoperability between modules. The topics covered in this paper are:

- Kinds of principle products
- Assembly and breakdown structures
- Kinds of principle activities and methods
- Resources, view or nature of something
- The ARM – MIM gap
- External classification or entity subtyping

2. Kinds of principle products

The purpose of this clause is to analyse the kinds of product in STEP modules today and to

- identify the basic ones
- identify which kinds of products have to be addressed by product views only (product_definition)
- identify missing ones
- identify how to use classification and categorization of products

The overall objective is to come to an agreed consensus of the kinds of products needed through STEP with the focus on STEP modules.

2.1. Product subtype tree today

As of today we have the following hierarchy of subtype of product:

- Product
  - Attachment_slot (239)
  - Breakdown (236, 239)
    - Functional_breakdown
    - Hybrid_breakdown
    - Physical_breakdown
    - System_breakdown
    - Work_breakdown_structure
    - Zone_breakdown
  - Breakdown_element (236, 239)
    - Functional_element
    - Physical_element
    - System_element
    - Work_breakdown_element
    - Zone_element
  - Document (all)
    - Physical_model (236)
    - Software (210)
      - Analytical_model (210)
      - Rule_product (210)
  - Envelope (239, MIM subtype)
  - Function_based_behavior (233)
  - Information_packet (233)
  - Input_output (233)
  - Interface_connector (239)
  - Interface_specification (239)
  - Justification (239, MIM subtype)
  - Message (239, MIM subtype)
  - Network_function (210)
  - Part (All)
  - Product_as_individual (239)
  - Requirement (210, 233, 239)
  - Se_breakdown (233)
  - Substance (210)
    - Chemical_compound
    - Chemical_element
    - Physical_particle
  - System_element (233)
  - Template (210)
  - Transformation (233)

The products indicated with “MIM subtype” are only products on the MIM level, but not on the ARM. Not all products are already balloted and published.

The impression is that this list is somehow arbitrarily. It is not possible to identify an overall structure. It is not clear which kind of product to use when and how.

In this paper we want to show that not every product listed here is a real product but is only a view to some products. Also the structure should be re-arranged to achieve consistency throughout all APs. Only when APs talk on the same kinds of products they can interoperate with each other.

2.2. About Product

The entity product is defined in two places, for the IR/MIM in the Integrated resource part41 and for the ARM in module ISO/TS 10303-1017:2004 Product identification. The definition in both is similar enough.

The definition of product in module 1017 says:

> *A Product is the identification of a product or of a type of product. It is a collector of data common to all revisions of the Product.*
>
> *NOTE 1 Products that this entity data type can represent, include:*
>
> - *products existing in the real world;*
> - *products that may come into existence as a consequence of some realization process. This includes parts and documents;*
> - *products that are functions.*
>
> *In the interpreted models, these various meanings are represented within instances of the entity product_related_product_category, with prescribed values of the category name. For example, the category name 'document' shall be used when characterizing the fact that a product is actually a document.*
>
> *EXAMPLE 1 The SS Titanic is a product that could be represented by the entity data type Product.*
>
> *EXAMPLE 2 Lifeboat is a class of products that could be represented by the entity data type Product. Each lifeboat on the SS Titanic is a member of this class.*
>
> *NOTE 2 A product is identified by an organization or a person in an organization. The definition of the domain of uniqueness and the mechanism for guaranteeing the uniqueness of product id are outside the scope of this application module.*
>
> *NOTE 3 A product may have zero or more versions. A version of a product is represented with an instance of the entity Product_version or of one of its specializations*

Issue:
From the above description it is clear that product and product_version are abstract concepts. Without further specialization it is not possible to do any data exchange or integration because we would not know what it means. We can’t make it a topic for APs to decide whether or not to allow instances of the supertype Product (without any categorization).
Proposal: make entity Product and Product_version abstract.

2.3. About Product_category

The definition in module says:

> *A Product_category is a type of product that is defined for a purpose that is specific to a module or application protocol.*
>
> *NOTE For the purpose of a general classification of products, use entity Class and Classification_assignment, standardized in ISO 10303-1070 and ISO 10303-1114.*

And is used in a typical subtype of product like this:

```
ENTITY Part
  SUBTYPE OF (Product);
WHERE
  WR1: SIZEOF(['part', 'raw material', 'tool']*types_of_product(SELF))=1;
END_ENTITY;
```

Issue:
Product_category is not needed at all on the ARM level because it’s functionality is already covered by subtypes of Product. Product_category on the ARM is a superfluous complication on the ARM level.

Proposal: Remove module Product_category completely, and also entity Product_category_assignment and function types_of_product.

2.4. Basic kinds of product

Looking to the real nature of products there are only tree basic kinds but with further specialization:

The **individual product**. This is the physical product we can touch and which we can mark somehow as individual, e.g. by applying a serial number. The Eifel tour in Paris is an example of an individual product (entity Product_as_individual). A main characteristic of individual products is that they are all somehow different, even when based on the same design. E.g. you may have 2 identical pens in your hand, but when you start measure the dimensions in very detail you will figure out that they are somehow different due to the tolerances of every fabrication process.

The **typical product** which may be physically realized one or several times. Since practically every design can be realized more than once a design is in all cases a typical product. The design of the Eifel tour in Paris is an example of a typical product. It may have been build more than once.

The **information product** is the last basic kind of products. Like a typical product it may be the result of a design process, but it is not possible to realize the information physically – there may be only a physical carrier for the information like a CD or a book which are all typical product.

Physical_model is not a Document but a Part.

2.5. More on typical products

Part is the most prominent representative of a typical product. AP214 defines a Part_version (Item_version) as something we can buy or build. From this we can easily deduce that a Part_version can't have any variants or undefined parameters - it is fully specified. Otherwise you can't buy or build it.

AP210 introduced **Template**, a concept similar to part. The main difference is that it is not possible to individually buy or build a template. A Template shows only up as an occurrence within a Part (assembly_definition).

Chemical substance is another new typical product. It represents an idealized "clean" kind of product, which does not exist in reality, and so it is not possible to buy or build it. But Chemical substances may be composed of other substances with new characteristi. part may be made out of some substance.

What is missing from the list here is a **Variant Part**, one with open parameters. E.g. a shirt in some size, a microprocessor in 500, 600 or 700 MHz or a car with the 60, 80 or 100 horse-power engine. It is clear that we can buy or build the Variant Part once the parameters are specified. In current module we would need to use Physical_breakdown for this, but this is very imprecise since we don't know what this really is. And Physical_breakdown does not allow to specify an assembly_structure, something we clearly need to do (here much more is needed to say ...TBD)

2.6. More on information products

A **Document** is information - no question, but a **Message** is well. Since we don't really know what a message differs from Document we should make it at least a subtype of Document.

A common understanding is that a document has information with meaning to a human user. A message may be addressed to a human user - or to some engine.

Software is written by humans - but can be executed after compilation by a machine. An Analytical_model is used in a similar way like software, but it has a specific ports and parameters to interact with other models or the ...

2.7. About breakdown

This is one of the most problematic areas in the STEP Application Modules. They are derived somehow from the AP214 ARM concepts complex_product and product_constituent, both map to product_definition. AP214 left it open to what kind of product they belong - on the ARM complex_product and product_constituent are stand-alone objects. Another source is the Nato Product Data Model NPDM (formerly called Nato CALS Data Model, NCDM). Here breakdown is also just a specialization of product_definition. We have to clarify for what it is good for to have Product_breakdown and Product_breakdown_element and all the subtypes on the product level (instead on the product_definition level only).

The more I think on this the more clear the answer is to me. These things should not exist on the product level at all. In all examples I know about or I can imagine a Breakdown and Breakdown_element is just a view of one of the products listed above, in most cases a specialized view for a Part/_version.

(This discussion needs to be extended further on for more problems on Breakdown_context, Breakdown_element_usage, Breakdown_of. TBD)

Note that there is also a module and entity on Product_group, another concept totally overlapping. Cleanup needed!

2.8. What are no products at all

The concept of a Hardcopy is very problematic because it is not clear whether it is an individual or a typical thing. I strongly recommend to remove hardcopy from modules completely (no deprecate). Modules are on a TS level and when moved to the IS we can do such a thing. For AP214 hardcopy will probably continue to exist longer.

2.9. New proposed subtype tree of products

After performing all the changes discussed in this paper the new structure of product subtypes would look like this:

- Product (Abstract)
  - Product_as_individual
  - Information_product (Abstract, new)
    - Document (all)
      - Message
      - Envelope (or better view only)
      - Justification
      - Information_packet (or better view only)
    - Requirement (210, 233, 239)

        - Software (210)
            - Analytical_model (210)
            - Rule_product (210)
        - Network_function (210)
    - Typical_product (Abstract, new)
        - Part
            - Physical_model (236)
        - Variant_part (new)
        - Template (210)
        - Substance (210)
            - Chemical_compound
            - Chemical_element
            - Physical_particle

Products to be covered by Product_view_definitions only!

- Attachment_slot
- Breakdown
    - Functional_breakdown
    - Hybrid_breakdown
    - Physical_breakdown
    - System_breakdown
    - Work_breakdown_structure
    - Zone_breakdown
- Breakdown_element
    - Functional_element
    - Physical_element
    - System_element
    - Work_breakdown_element
    - Zone_element
- Function_based_behavior
- Interface_connector
- Interface_specification

Entities which should not be a subtype of Product or Product_view_definition at all

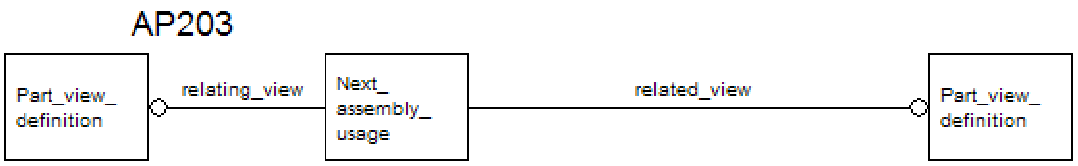
- Input_output
- Se_breakdown (what is this?)
- Transformation

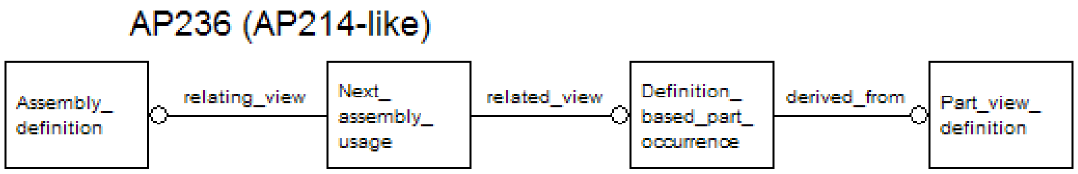
3. Assembly structures

Traditionally we have three different kinds of assembly structures in STEP (on the AIM level):

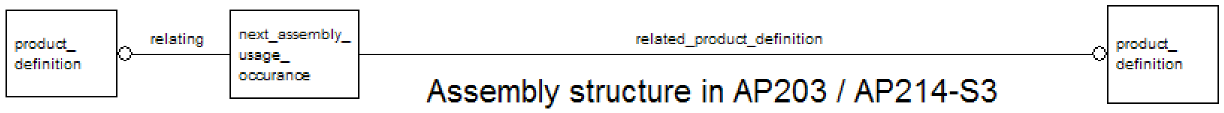
- AP203 and AP214-UOF-S3
- AP210
- AP214-UOF-S7

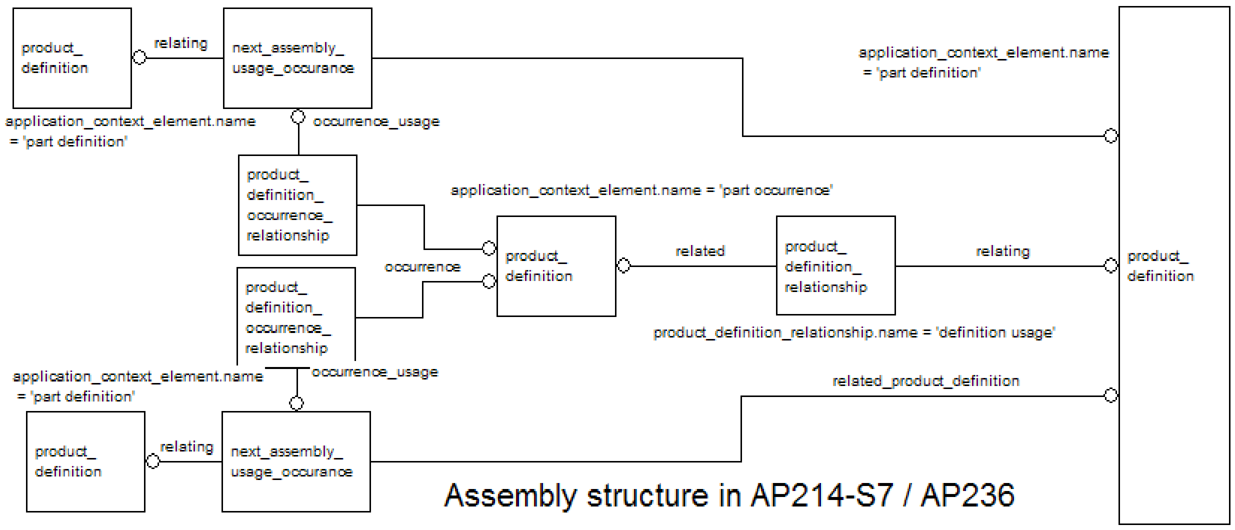
These different structures are reflected in the three alternative mapping of Product_occurrence. Here a graphical representation of the differences (a kind of instance diagram) on the ARM level:

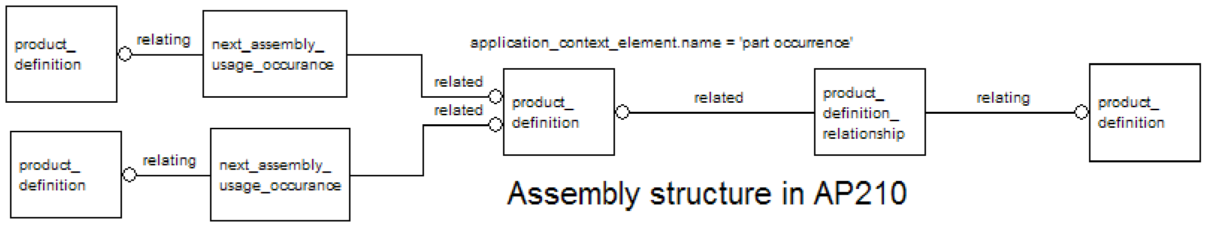




Note that on the ARM level AP236 and 210 are almost identical (for the supertypes), but on the AIM level they differ:


product_definition
relating
next_assembly_usage_occurance
related_product_definition
product_definition
Assembly structure in AP203 / AP214-S3


product_definition
relating
next_assembly_usage_occurance
application_context_element.name = 'part definition'
application_context_element.name = 'part definition'
occurrence_usage
product_definition_occurrence_relationship
application_context_element.name = 'part occurrence'
product_definition
related
product_definition_relationship
relating
product_definition
occurrence
product_definition_occurrence_relationship
product_definition_relationship.name = 'definition usage'
application_context_element.name = 'part definition'
occurrence_usage
related_product_definition
product_definition
relating
next_assembly_usage_occurance
Assembly structure in AP214-S7 / AP236

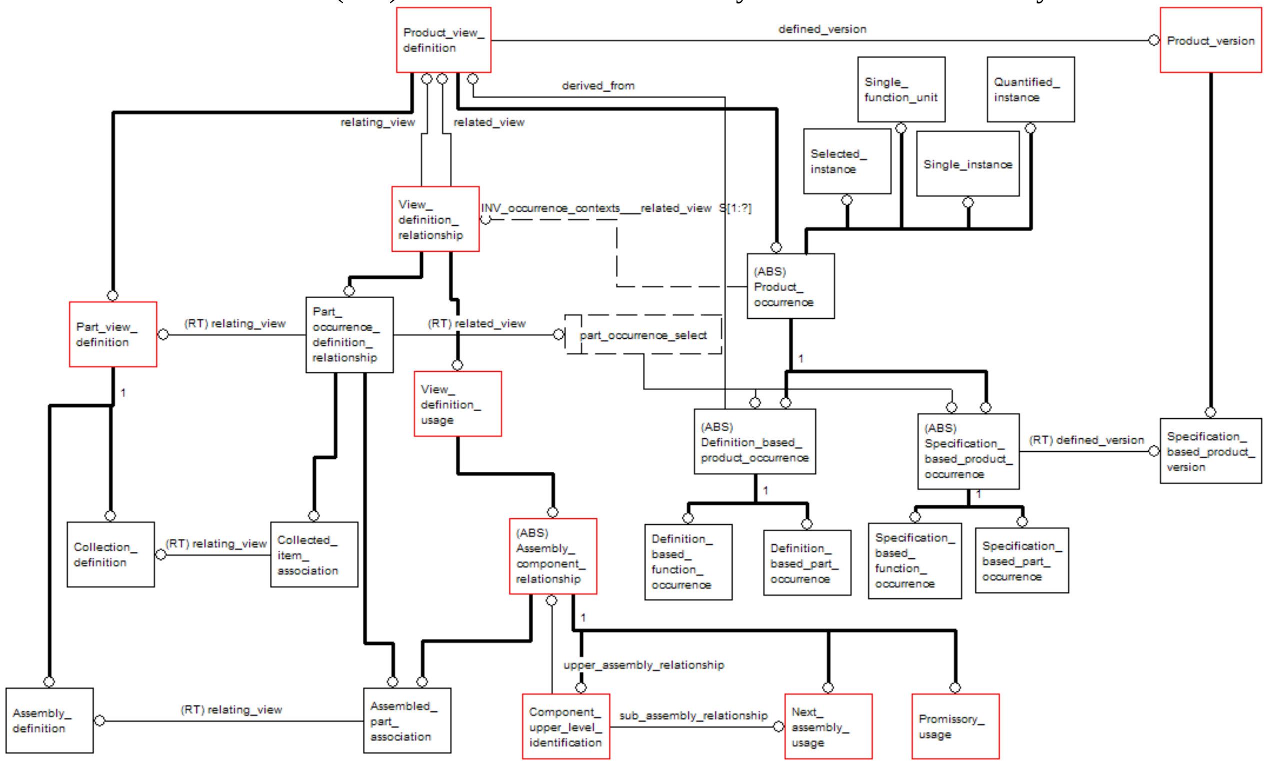

product_definition
relating
next_assembly_usage_occurance
application_context_element.name = 'part occurrence'
related
related
product_definition
related
product_definition_relationship
relating
product_definition
product_definition
relating
next_assembly_usage_occurance
Assembly structure in AP210

The old AP203 structure is clearly insufficient because it does not support an independent "part_occurrence" concept like AP210 and 214 have. In the ARM of AP214 we have only one data-model for the assembly-structure but two alternative mappings, S3 and S7. S3 is somehow a subset of the S7 mapping, covering the AP203 case. For AP210 the complicated S7 trick was not acceptable because this would have lead to many problems - hear a clean ARM-AIM mapping was essential. AP236 used the AP214 ARM assembly structure and converted it to a modular level. As a result AP210 and AP236 are today compatible on the ARM level, but not on the AIM level. AP203-ed2 is on the ARM level neither compatible with AP210 nor with AP236.

So how to go on with this jumble? - only a clear and explicit long-term strategy can help. I recommend to

- include Part_occurrence in for AP203ed2/3 to so that it becomes compatible on the ARM level.
- Enforce any other upcoming AP to use part_occurrence for assembly structures
- keep all three alternative mappings valid for data exchange for the particular APs (as is today)
- Make the AP210 mapping the recommended mapping for data integration in a STEP-database because it can absorb all AP203 and 214 cases but keep the clean occurrence structure.

The following picture indicates the AP203 ARM entities today in red and the additional AP236 (214) entities in black color. Many of them are also used by AP210.


Product_view_definition
defined_version
Product_version
derived_from
Single_function_unit
Quantified_instance
relating_view
related_view
Selected_instance
Single_instance
View_definition_relationship
INV_occurrence_contexts__related_view S[1:?]
(ABS) Product_occurrence
Part_view_definition
(RT) relating_view
Part_occurrence_definition_relationship
(RT) related_view
part_occurrence_select
1
1
View_definition_usage
(ABS) Definition_based_product_occurrence
(ABS) Specification_based_product_occurrence
(RT) defined_version
Specification_based_product_version
1
1
Collection_definition
(RT) relating_view
Collected_item_association
(ABS) Assembly_component_relationship
Definition_based_function_occurrence
Definition_based_part_occurrence
Specification_based_function_occurrence
Specification_based_part_occurrence
1
upper_assembly_relationship
Assembly_definition
(RT) relating_view
Assembled_part_association
Component_upper_level_identification
sub_assembly_relationship
Next_assembly_usage
Promissory_usage

Several issues can be read out of this diagram:

- Assembled_part_association and Collected_item_association must be ONEOF
- View_definition_relationship must be abstract
- Part_occurrence_definition_relationship must be abstract and be a subtype of View_definiiton_usage

- The inverse relation on Product_occurrence must be turned into a constraint saying that every usage from View_definition_usage must be Part_occurrence_definition_relationship
- Assembled_part_association must be in an AND constraint with ONEOF (Component_upper_level_indication, Next-assembly_usage or Promisory_usage)
- The two big subtype-groups of Product_occurrence must be in some AND combination, e.g.
- a Definition_based_part_occureence must be combined with either Single_instance or Quantified_instance.

3.1. Variant assembly structures

Variant assembly structures are in the scope of AP214 and AP236

4. Kinds of principle activities and methods

As of today we have the following subtypes of Activity and Activity_method:

- Activity
  - Activity_actual
  - Design_object_management_relationship
  - Directed_activity
  - Observation (indirectly via MIM)
  - Work_output (indirectly via MIM)

- Activity_method (ARM)
  - Scheme
  - Scheme_entry
  - Scheme_version
  - Task_element
    - End_task
    - Exit_loop
    - Structured_task_element
      - Concurrent_elements
        - Simultaneous_elements
      - Decision_point
      - Looping_element
        - Repeat_count
        - Repeat_until
        - Repeat_while
      - Task_element_sequence
    - Task_element_levels
    - Task_invocation
    - Task_step
      - Advisory_task_step
  - Task_method
  - Task_method_version
  - Condition (indirectly via MIM)

    - Condition_evaluation (indirectly via MIM)
    - Information_right (indirectly via MIM)
    - Information_usage_right (indirectly via MIM)

A principle question about Activity, Activity_method and all their subtypes is about their nature in respect to identity. These questions are very important when the goal is not only to exchange a particular snapshot of the actual state but also to record the history:

- what happens when we need a new version of an Activity/_method, e.g. when we need to change some attribute like time, duration, etc. Do we create a new instance of Activity/_method and establish some identity-relation between them? - or do be keep the only one instance and record the details on the assigned attributes?
- If we create a new instance, what happens with the old relations? - do we need to duplicate them anew or is it the responsibility of some application to figure this out. If yes along which rules?
- what happens when we change the choosen_method attribute of an Activity? In this case we must create a new instance and relate it to the previous one in some version-relationship

Further unanswered questions are

- can Activity_methods be related with absolute times, e.g. planed start date 2005_06_06? I would expect that this make only sense for Activities, but not for Actitivy_methods.
- What is the generic method to say that one Activity_method is further detailed in another Activity_method (or subtypes). Today we have the overlapping entities Task_invocation and Activity_method_realization. Which is doing what? (I guess one has to be removed).
- Can we include the same Activity/_method in several higher level groupings? E.g. can an Activity we assigned to more than one Project? Can a Scheme_entry be in the contents of more than one Scheme_version? Can a Task_element be in the contents of more than one Task_version?

Activity_actual is a questionable specialization of Activity. According to AP214 an Activity is actual if it has an "actual start-date".

It is hard to understand that Work_output is an activity. Like for Resource_item (see below) this is only a view to something, or better to say a relation between a result (the product) and some activitiy in the role "work-output". This needs also harmonization with AP214 Process_operation_input_or_output.

There is no clear separation between Task_xxx and Scheme_xxx. Both structures are similar but when to use which? It may be the case that Task_xx should better become subtypes of the corresponding Scheme_xxx.

It is easy to understand that Condition is an Activity_method and so it should be a formal subtype of it. But what about Infromation_right this looks more like a specialization of Independent_property.

The result of a detailed review of all subtypes of Activity and Activity_method must be a clear definitional separation so that implementations know which one to choose in which case. For this also the question on typical usages (what happen when) listed above must be normatively answered.

5. Resources, view or nature of something

This clause is very draft for now.

There are problems with the modules:

- ISO/TS 10303-1268:2004 Resource item
- ISO/TS 10303-1267:2004 Required resource
- ISO/TS 10303-1266:2004 Resource management
- ISO/TS 10303-1269:2004 Resource as realized

It is not clear what a resource_item and all the stuff on top of it is really. It seems that resource_item is only a specific view or better to say role of something. If this is the case then I wonder why this entity has a "name" attribute and why it can collect several other resource_items (the attribute). Maybe the whole entity resource_item should be replaced by resource_item_select.

Overall examples are needed to figure out how this whole thing should work.

6. Architectural problems

6.1. The ARM – MIM gap

Let us remember that the very first APs had no formal ARM data model (Express or IDEF1X). Mapping tables had only the purpose to map application concepts to the integrated resources. Later on ARM Express models get more and more detailed till the point that some people say that they only want to implement ARM and no longer the AIM. This is because a lot of the complexity was moved from the AIM/MIM/IR level to the ARM. Unfortunately the structure of Application Modules intensifies this process to such a degree that many generic IR concepts are now reflected to the ARM level. As a result the ARM of the application modules got very complex and low level. But it is not complete - it misses many essential rules and semantic details.

As an example take the ARM entity "Product" (part 1017) and compare it with the IR entity "product" (part 41). On the ARM it is very vague what is meant with "id" and "name". In the IR however it is clearly defined that id is of type identifier and name of type label. Both types have String as the underlying type, but they have very different semantic meaning. Representation_item is another interesting candidate; it misses the important rule WR1.

It is a matter of fact that many of the basic modules are a one to one copy of concept already standardized in the common resources. This process is really questionable, but it is dangerous when while copying not acceptable simplifications are made. The objective of introducing application modules is to cover application concepts on a higher level, not to compete with common resources.

While the ARM evolved in being more and more low level we can also observe that people develop very tool or application specific data models and try to map them on existing ARM models. The German specification for gears and the ProSTEP KBL model for cable-harness are examples. They define what we could call an ARM2 model which is then mapped to an ARM1 level (of AP214 and 212 in these examples). And since the ARM1 is too generic they prefer to implement on the ARM2 level. We can foresee that this is a recursive process and that next people come with an ARM3 model and map it to an ARM2 etc. It is clear that this is in no way a clean recursion. It would be much better to introduce more and more specialized application models and to make the more generic concepts abstract for a specific usage.

6.2. XIM, a merger of ARM and AIM

It is possible to close the gap between ARM and MIM by merging the separated models into one integrated data model where the ARM schema is turned into a specialization of the corresponding MIM schema, called XIM (extended Integrated Model). XIMs allow implementing STEP on the ARM level and then to transfer the data to the MIM level and back without any loss of information. XIM and the underlying software taking care of the mapping tables are a proprietary technology of LKSoft.

There is a potential way to adopt the XIM concept for standard application modules if SC4 would decide to go for this and if the needed resources would become available. With a few changes and enhancements in the implementation methods (part 11, 21, 22, 28, ...) the application modules could be simplified drastically. As a result the modules would have only one schema (instead of two) and no mapping table. The single Express schema would be directly based on integrated resources (like the MIMs today) and in addition covering all ARM concepts explicitly.

The main needed extensions of Express would be the introduction of the concept of a CONNOTATIOAL SUBTYPE (see draft of disbanded Express edition 3). The keyword CONNOTATIONAL allows to define subtypes of entities not visible in an exchange file.

Another needed extension of Express would be a simplified way to write where rules. Note that constraints available in mapping specifications are much easier to read, write and verify than corresponding traditional where rules, using operators such as QUERY, USEDIN and TYPEOF. Many years of experiences of writing, reviewing and correcting mapping constraints and rules for AP210, 214 and other APs shows that the amount of time differs by a factor of 10 or so.
Mapping tables are not easy to read and write, but they are trivial in comparison to equivalent where rules.

6.3. Module hierarchy

One of the main principles of the modular architecture is that a lower-level module is used either completely by a higher-level module or not at all. The word "use" is translated into the Express construct USE FROM of the whole lower-level schema for both, ARM and MIM. The effect of USE FROM is that everything defined in the lower-level schema is made available in the upper-level schema as if it is defined there.

In Express we also have REFERENCE FROM which has a much more restricted meaning and both constructs could be used partially for only a few selected data types. But this is not permitted in the current philosophy, enforcing STEP modules to be use like Lego-stones, in theory. In practice we have to fight with a couple of problems:

6.3.1. Scope statement

The intention of including a lower-level module into a higher-level module is that everything from the lower-level module becomes part of the higher-level module. Is this really the case?

From a formal point of view the answer is NO!
This is because with USE FROM we include only the ARM and MIM, but not the

- scope (and introduction)
- mapping specification,
- application module implementation and usage guide
- Technical discussions

There are further problems because scope statements shall be unique for all step standards and parts. How can this be the case if we USE FROM a lower level module and adding only a few additional things to it, e.g. by extending a select.

The only clean solution to this problem is to formally include the Scope and everything else from a lower module into the higher level module by a statement like this:
"–items within the scope of application module Xxx, ISO/TS 10303-1xxx; "

There is only one exception when this should not be the case. This is when a SUBTYPE CONSTRAINT is introduced to make an entity from a lower level ABSTRACT. A similar effect can be achieved by where rules.

6.3.2. Cyclic dependencies

In some case we have the situation that module A is using module B and module B is using module A. It could also be more complex such as A is using B, B is using C and C is using A. It is clear that in such cases the Lego idea is broken - a stone cannot be simultaneously be above and below another one.

Within the AP210 project we have developed a simple tool to detected such cyclic dependencies. With the help of this tool we detected a few such dependencies in non-AP210 modules. All cyclic dependencies in AP210 modules got resolved except of two cases where it was simply not possible to do something under the current rules.

6.3.3. Subtype Constraint and Rules

It is clear that a Part and a Product_individual are exclusive to each other. Since both are subtypes of Product we have to write a subtype constraint. AP239 has this rule (ap239_prdi_restrict_product_subtypes); AP236 does not. And when we analyse everything in more detail we will see that all the APs adds their specific rules and subtype constraints in the Implementation modules and all do it differently. There is no way to align this process and to ensure that all the APs are doing it the same way.

This situation is not acceptable. A solution would be to have a common higher level module between AP236 and 239 with this rule. But this is impracticable for several reasons. One is that AP236 supports other kinds of Products than AP239 does.

An easy solution would be to allow selective REFERENCE FROMs to other modules, even from lower-level modules to higher-level modules. This does not mean that the referenced module becomes part of the other level module. But it allows to write subtype constraints and other rules in a common place for the case that such two modules are used together on a higher level.

The given example could then be addressed like this:

```
SCHEMA Product_identification_arm;
   REFERENCE FROM Part_and_version_identification_arm (Part);
  REFERENCE FROM Product_as_individual_arm (Product_individual);
 ...
ENTITY Product
  ABSTRACT SUPERTYPE
  SUBTYPE OF (ONEOF (Part, Product_individual, ...));
  id : STRING;
  name : OPTIONAL STRING;
  description : OPTIONAL STRING;
END_ENTITY;
...
END_SCHEMA;
```

AP203 and AP210 who does not include the module Product_as_individual would simply not see the entity Product_as_individual, but both AP239 and AP236 would see it.

Also the problems mentioned about on cyclic dependencies and other more complex problems could be much better addressed.

Express provides us with a good functionality to address the Lego approach and simultaneously to write the needed constraints. Why not to use this functionality?

6.4. Conformance classes

We need Options for Conformance-classes to avoid inflation. TBD

6.5. Extended selects

If often happens that the same base-select type is extended more than one by the same item. This is not good. This has to be avoided. TBD

6.6. Similar higher level modules

We have too many higher level modules with almost the same USE FROMs. This is not good because it is a duplication which can later lead to many problems. We have to turn stepmod into a clear structure.

Example:
  Schema X, USE FROM A, B
  Schema Y, USE FROM A, B, C
shall be converted into
  Schema X, USE FROM A, B
  Schema Y, USE FROM X, C

TBD

6.7. Rules in general

Many Express rules and subtype constraints are missing today, both on ARM and MIM. Without much stricter constraint data models, data exchange will not work.

TBD.

6.8. Express long froms

Express long froms are an absolutely useless concept. Not needed at all from a technical point of view. How long SC4 want to deal this this?

It could be useful to provide a long from for the top implementation modules of the 4xx series. But for implementation modules form the 1xxx series it is not even useless but even dangerous.

TBD

7. External classification versus entity sub-typing

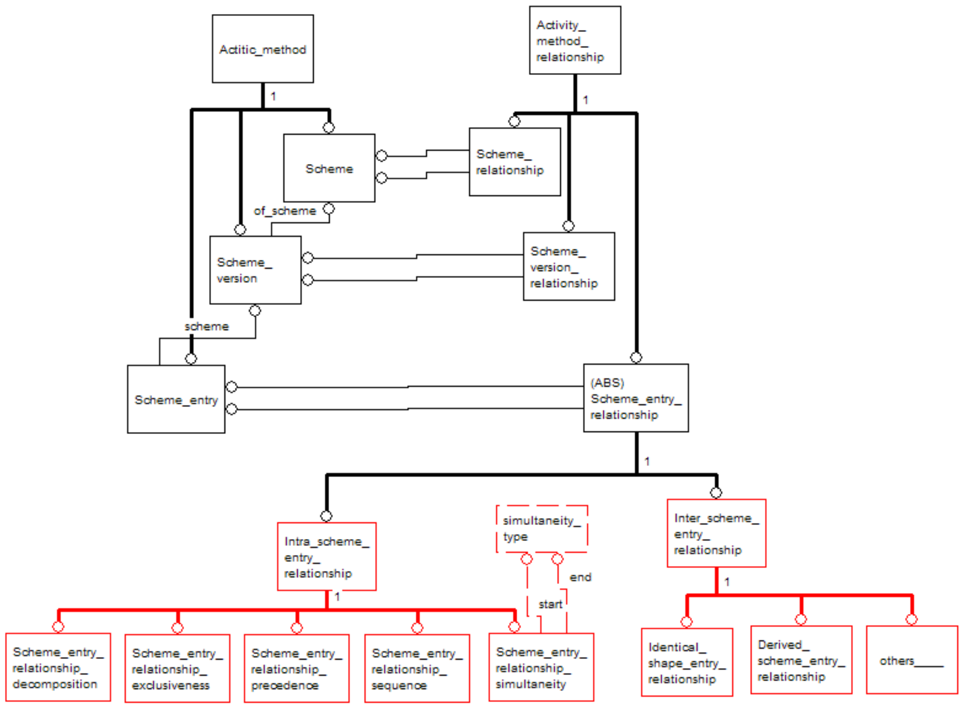
There is an ongoing debate on whether we should use external classification according to some dictionary or to use sub-typing of entities. Both ways have advantages and disadvantages.

We can observe consensus within SC4 that the specialization of e.g. Product into Part and Document is done via entity sub-typing. On the other hand we think that SC4 has a common understanding that the specialization of e.g. Part into nut and bold is outside the scope of ISO 10303 and should be done in a library. But here are several areas where we have not yet consensus. As a compromise I would like to propose the following guidelines:

1) APs are free to extend classification_item for all kinds on entities when they can show that this makes sense in some areas. It is clear that during data exchange all such references to external libraries may get lost when the receiving system is not aligned with the sending system for the same library.
2) All generic xxx_relationships and xxx_assignments (on the ARM level) shall be by default abstract because they are so generic that a receiving system can not do anything useful. Specific non-abstract subtypes classification_defined_xxx_relationship will be introduced for those relationships who's meaning is only defined in some external library or by a user specific classification system.
3) APs have already defined many specific kinds or xxx_relationships such as "sequence", "decomposition", "alternate" etc. For those we will introduce specific subtypes of xxx_relationship and xxx_assignments with pre-defined meaning. This enables basic data exchange between different implementation without the need to agree on other standards.

In module
ISO/CD-TS 10303-1760: Pre defined product data management specialisations
several missing specializations are defined. Most of them should be moved into the modules there the supertypes are originally defined. Only then we can hope that data exchange will work.

For the example of Scheme_entry_relationship the result may look like this:


Actitic_method
Activity_ method_ relationship
Scheme
Scheme_ relationship
of_scheme
Scheme_ version
Scheme_ version_ relationship
scheme
Scheme_entry
(ABS) Scheme_entry_ relationship
Intra_scheme_ entry_ relationship
simultaneity_ type
end
start
Inter_scheme_ entry_ relationship
Scheme_entry_ relationship_ decomposition
Scheme_entry_ relationship_ exclusiveness
Scheme_entry_ relationship_ precedence
Scheme_entry_ relationship_ sequence
Scheme_entry_ relationship_ simultaneity
Identical_ shape_entry_ relationship
Derived_ scheme_entry_ relationship
others_____
1

(See also the discussion in clause 4 on this)