**DDI Collections (Proposal)**

This document defines a generic collection structure to support managed and unmanaged collections in DDI 4 containing both unique and non-unique members. Such a generic structure can be used to model different types of groupings, from simple unordered sets to all sorts of hierarchies, nesting and ordered sets/bags. In addition, they can be extended with richer semantics (e.g. generic, partitive, and instance, among others) to support a variety of DDI 3.2 and GSIM structures, such as Node Sets, Schemes, Groups, sequences of Process Steps, etc.

**Formal framework**

**Definition 1:** A collection **C** is the tuple **(S, ≤)** where **S** is a container (set or bag) and “**≤**” is an order relation. “**≤**” could be empty, in which case the container is unordered.

**Definition 2:** An order relation “**≤**” is a [binary relation](http://en.wikipedia.org/wiki/Binary_relation) over a container (set or bag) **S** that is [*reflexive*](http://en.wikipedia.org/wiki/Reflexive_relation), [*antisymmetric*](http://en.wikipedia.org/wiki/Antisymmetric_relation), and [*transitive*](http://en.wikipedia.org/wiki/Transitive_relation). In other words, for all members **a**, **b**, and **c** in **S** we have:

* **a ≤ a** ([*reflexivity*](http://en.wikipedia.org/wiki/Reflexive_relation));
* if **a ≤ b** and **b ≤ a** then **a = b** (*[antisymmetry](http://en.wikipedia.org/wiki/Antisymmetric_relation" \o "Antisymmetric relation)*);
* if **a ≤ b** and **b ≤ c** then **a ≤ c** ([*transitivity*](http://en.wikipedia.org/wiki/Transitive_relation)).

The order relation “**≤**” can be *total* or *partial*. It is sometimes called “*precedence*” because it represents an order of precedence within the set. When every pair of members from **S** is in the order relation “**≤**” we say it is total; otherwise it is partial. A total order relation, also called linear order, defines a sequence.

The right-hand side of Figure 1 shows one possible UML representation of a collection **(S, ≤)**. The **Collection** class represents the container **S** and the **Member** class its elements. An optional **orderRelation**, reified as an association class to allow for subtyping, provides the predecessor-successor pairs in the order relation **≤**, which is optional. The **type** attribute in **Collection** indicates whether the container **S** is a *bag* or a *set*, and in **Member** whether the order relation is *total* or *partial*. The left-hand side of the figure shows an example of a **NodeSet** modelled as a specialization of the generic collection structure on the right-hand side. Note that multiple order relations are possible for the same container **S**, e.g. the **parent-child** and **part-whole** shown.

Note that the predecessor-successor pairs model only *direct precedence*, which is the minimal information required to reconstruct the whole order relation by adding reflexivity and applying transitivity.

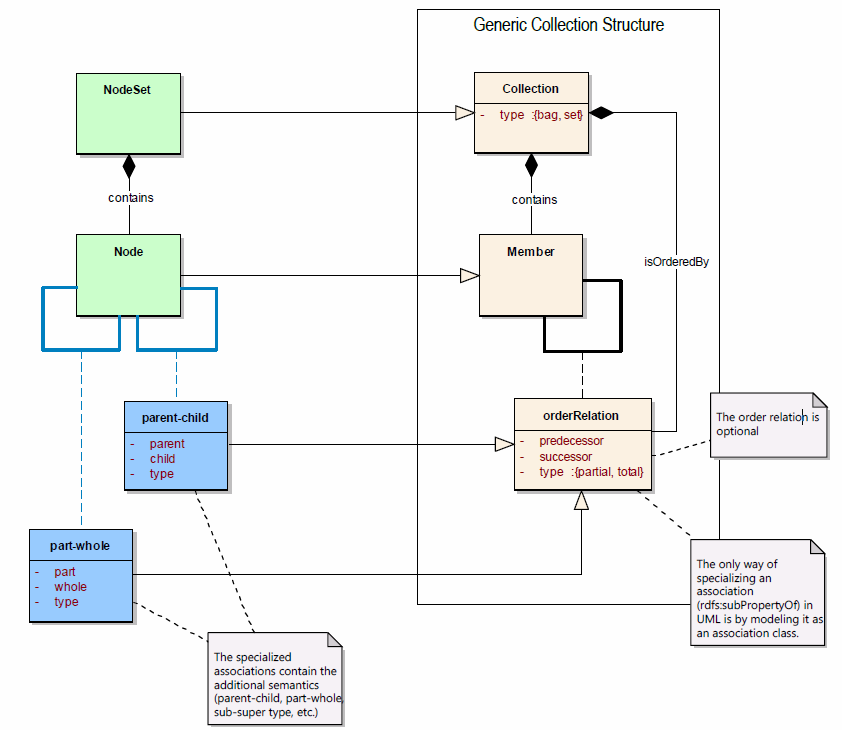


Figure 1: NodeSet as a Collection with specialized order relations

**Modelling classifications hierarchies with order relations**

As an example, we will show next how ordered sets can be used to model the different types of hierarchies and nesting that appear in **Classifications**.

**Example 1**: Let us define a collection **(L, prec-L)** where **L := {Sectors, Subsectors, Industry Groups, Industries, National Industries}** is the set of classification levels in NAICS 2012. The levels in **L** belong to a hierarchy that can be defined by the total order relation **prec-L := {(Sectors, Subsectors), (Subsectors, Industry Groups), (Industry Groups, Industries), (Industries, National Industries), (Sectors, Industry Groups), …}**. Figure 2 shows a visual representation of **prec-L**.



Figure 2: Total order of Levels

**Sectors** is the top level and **National Industries** the bottom. It is easy to see that the relation is total because all members are related to each other. Red arrows indicate direct precedence in **prec-L** whereas grey arrows indicate indirect precedence implied by transitivity (reflexivity is not shown). In other words, red arrows are *parent-child* relationships between levels and grey arrows are *ancestor-descendant* ones. For simplicity, we will represent only direct precedence in the following examples.

A *total* order relation can define only sequences, like the one in Example 1. To define richer structures, like the hierarchies of members and levels in a classification, we need a *partial* order relation.

**Example 2**: We will define next a collection **(I, prec-I)** where **I := {Agriculture etc., Crop Production, Fishing etc., Fishing, ..., Manufacturing, Food, Chemical, Grain, Flour Milling, …}** is the set of classification items in NAICS 2012. The items in **I** also belong to a hierarchy that can be defined by the *partial* order relation **prec-I := {(Agriculture etc., Crop Production), (Agriculture etc., Fishing etc.), (Fishing etc., Fishing), …, (Manufacturing, Food), (Manufacturing, Chemical), (Food, Grain), (Grain, Flour Milling), …}.** (Names were abbreviated for simplicity.) We used NAICS category names to identify the classification items because they are more illustrative than their codes (although they are not unique). However, we need to keep in mind that **prec-I** contains whole classification items rather than just their category names (or codes). Figure 3 shows a visual representation of a fragment ofthe hierarchy of items.



Figure 3: Hierarchy of Classification Items

The blue arrows denote the *parent-child* relationships between items in the partial order relation **prec-I**. Note that the order relation is in fact partial, e.g. **Food** is not related to **Chemical**, **Chemical** is not related to **Animal** or **Grain**, etc. Such an order relation (with implicit transitivity and reflexivity) defines a number of trees, e.g. those rooted at **Agriculture etc.** and **Manufacturing**.

We have defined so far two of the building blocks to model a classification like NAICS: a collection **(L, prec-L)** consisting of a set of classification levels **L** and their hierarchy given by the *total* order relation **prec-L**, together with a collection **(I, prec-I)** consisting of a set of classification items **I** and their hierarchy given by the *partial* order relation **prec-I**. We need now to put them together by creating a mapping **M** (i.e. a binary relation) between items and levels. This mapping can be created in two different ways. One way is to group the items into exhaustive and non-overlapping sets (more formally, a *partition[[1]](#footnote-1)*), one for each level, and create the mapping between those sets and the levels. Figure 4 illustrates the result.



Figure 4: Hierarchies and mappings

Green arrows represent the mapping **M** between sets of items on the left and their respective levels on the right.

Alternatively, instead of defining the sets, we could simply map individual items to their levels, in which case there would be green arrows going from each item on the left to the levels on the right.

Additional associations to other classification objects (**Indexes**, **Maps**, etc.) can be modelled by mappings as well.

**In conclusion, all hierarchies and nesting that appear in classifications can be modeled by ordered sets.**

**A note on constraints:** There is an additional constraint in a classification that needs to be satisfied to guarantee that the hierarchies in both collections, i.e. items and levels, are consistent with each other. An informal definition of such a constraint, based on the figures above, is the following: *if there is a blue arrow from an item to another, there has to be a red arrow from the level of the former item to the level of the latter item*.

**Example 3**: Consider the last figure with classification items, level and mappings. There is a blue arrow from **Grain** to **Flour Milling**. Since **Grain** belongs to the **Industry Groups** level and **Flour Milling** to the **Industries** level (as indicated by the green arrows), in order for the classification to be well-formed there has to be a red arrow from **Industry Groups** to **Industries**, which is the case in the example.

This highlights the need for a constraint language at the conceptual level.

**Classification View in UML**

Figure 5 uses the Classification View, extended with order relations, as an example. For simplicity, the association classes of the order relations are not shown. Entities in green are those already present in DDI 4 whereas those in blue need to be added to complete the set of objects pertaining to Classifications (in alignment with GSIM).

A constraint on the aggregation indicates whether the order relation is *total* or *partial*. Cardinalities need to be consistent with the constraint, and thus with the type of order relation: 1/0..1 – 1/0..1 is compatible with total order whereas 1/0..1 – 1..\*/0..\* is compatible with partial order.

The self-aggregations in bold show order relations. Note that there is a partial order on **Nodes** (in blue) modeling the hierarchy of nodes within a **Node Set**, which is inherited by **Classification Item** within **Statistical Classification**. There is also a total order on **Levels** (in red), modeling the hierarchy of levels. Those correspond to the blue and red hierarchies in the NAICS example of Figure 4, respectively. The green mapping between nodes and levels in that figure is represented in the UML model by the *groups* association between the **Node** and **Level** classes.



Figure 5: Classification View with order relations

Other order relations can be defined as needed. For instance, if we want to order the nodes of each level in a sequence, we can just add the total order *precedes* on **Node**. That way, we will be able to make **Fishing** appear before **Hunting and Trapping**, **Flour Milling** before **Starch** and so on (see **Industries** level). The *isBasedOn* self aggregation **Statistical Classification** can also be viewed as a hierarchy and modelled with a partial order relation as shown.

Note that UML is not expressive enough to specify integrity constraints that need to be included. For instance, the constraint associated with the **Classification Item** class in the UML diagram is required to guarantee that the item and level hierarchies are consistent with each other. To specify such constraints we need a more expressive language to complement the UML model.

1. A *partition* of a [set](http://en.wikipedia.org/wiki/Set_%28mathematics%29) **S** is the union of non-overlapping and non-empty subsets of **S** [↑](#footnote-ref-1)