Basic Operations Over Models Containing Subset and Union Properties

Marcus Alanen and Ivan Porres

TUCS Turku Centre for Computer Science
Department of Information Technologies,
Åbo Akademi University
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
e-mail:{marcus.alanen,ivan.porres}@abo.fi

Abstract. The Meta Object Facility 2.0 and Unified Modeling Language 2.0 Infrastructure standards present novel metamodeling constructs called subset and union properties. However, they do not provide a complete definition of these constructs. This definition is necessary to construct modeling tools and to ensure their interoperability. In this article, we present the basic model operations over models containing subset and union properties. These operations are formalized using pre- and postconditions using substitutability as the main criterion for language specialization.
Keywords: subset and derived properties, metamodeling, MOF, UML

1 Introduction

The purpose of the Unified Modeling Language (UML) 2.0 Infrastructure [16] is to define the Meta Object Facility (MOF) 2.0 [15] and the UML 2.0 Superstructure [14]. It introduces several new concepts not present in MOF 1.x or UML 1.x, mainly: *subset* properties, *derived union* properties and property *redefinitions*. These concepts are useful to define a new modeling language as an extension of an existing one. Unfortunately, very little is told in [15, 16] about the actual meaning of these new constructs. This is a critical omission since these concepts are heavily used in the definition of the UML 2.0 Superstructure and are necessary to enable interoperability between software modeling tools, including model editors and model transformation tools.

In this article, we discuss the basic operations to edit models containing subsets and union properties and formalize them using pre- and postconditions. These basic operations are the elemental building blocks for a model repository supporting interactive model editors for UML and model transformation engines for languages such as QVT [13]. Although this article only presents a theoretical framework, we believe it contains important implications for the practical implementation of model repositories for UML 2.0.

We proceed as follows. Section 2 briefly presents a set-theoretic formalization of a metamodeling language that supports the new subset properties of MOF 2.0 and the UML 2.0 Infrastructure. The main contribution of the article is in Section 3, where the basic edit operations are discussed in detail. Finally, we discuss related work in Section 4 while Section 5 contains some concluding remarks.

2 A Simple Metamodeling Language

The main concepts used in metamodeling are classes and properties. A class represents a concept in a modeling language such as a UML Use Case or a Transition in a Statechart, while a property represents a feature of such a concept such as the fact that a Use Case has a name or a Transition has an event trigger.

As an example, the left part of Figure 1 shows a metamodel for a graph. This diagram shows two classes: *Vertex* and *Edge*, and four properties: *from*, *to*, *outgoing* and *incoming*. Each property has another property as its opposite. Together they define an association that is represented as a single line. In the example, we have the *from-outgoing* and the *to-incoming* associations. At the model layer, this bidirectionality means that when a *Vertex v* has an *Edge e* in its *outgoing* slot, the *Edge e* will have *Vertex v* in its *from* slot. The right side of Figure 1 shows an example model represented as an object diagram where each object is an instance of a class in the metamodel.

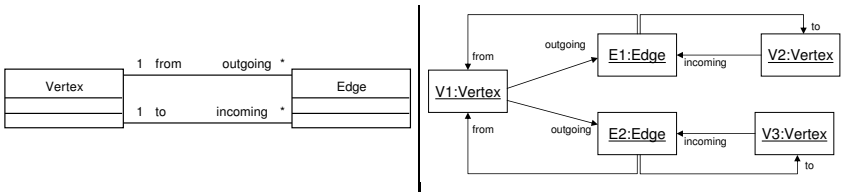

Fig. 1. (Left) Metamodel for a Graph; (Right) Example Model

MOF 2.0 also provides three main extension mechanisms for metamodels: class specialization, property subsets and unions, and property redefinitions. Class specialization is identical to class inheritance in object-oriented languages. A specialized class inherits all the properties of its base classes and can also define new properties. Subset and union properties are a mechanism to define the relationship between the properties in a specialized class and its base classes. Finally, property redefinition allows us to replace a property with another "compatible" one; however, compatibility is not precisely defined.

We can use specialization and subset properties to create a new metamodel in Figure 2 for a bipartite graph for our running example. The classes *Blue Vertex* and *Red Vertex* will now be specializations of *Vertex*. Also, the *fromRed* and *toBlue* properties will become subsets of the *from* and *to* properties, and similarly for the other properties. An example model is also shown in the figure. This metamodel is based on an example presented in [18]. The reader can find many complex examples of the use of subset and union properties in the UML 2.0 standards.

The intuition behind the metamodel is as follows: an element of type *Red Vertex* has four slots that correspond to properties *outgoing*, *incoming*, *outgoingRB* and *incomingBR*. Elements of type *Edge* can be inserted into the *outgoing* or *incoming* slot and elements of type *RedBlue Edge* can also be inserted into *outgoingRB*. At any mo-

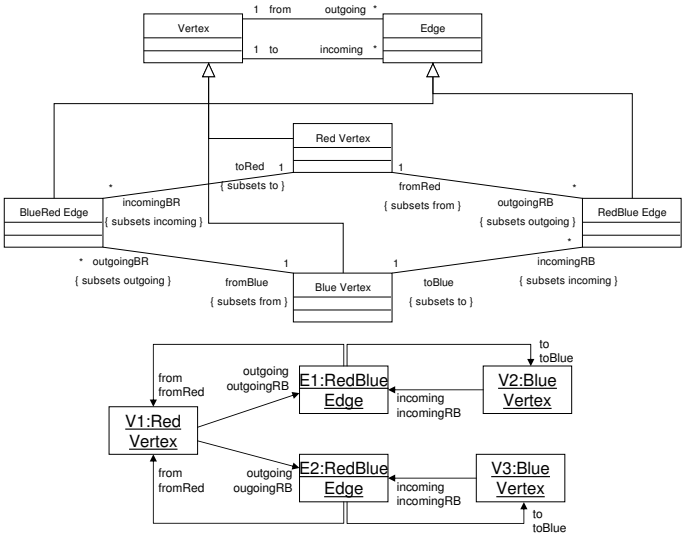

Fig. 2. (Top) Metamodel for a Bipartite Graph as an Extension of the Metamodel for a General Graph. (Bottom) Example Model for the Graph Metamodel

ment, the contents of the slot *outgoingRB* should be a subset of the contents of the slot *outgoing*.

The benefit of subsets in the running example is that graph traversal algorithms which worked on the initial metamodel in Fig. 1 should still work for bipartite graphs when using the metamodel in Fig. 2, and that if we only use elements from the bipartite graph metamodel, we can also be certain that the model describes a bipartite graph.

Our metamodeling language should support multiple inheritance since it is used extensively in MOF, as has been noticed by e.g. Anneke Kleppe [11]. Multiple inheritance forms very complicated inheritance hierarchies, among them the *diamond inheritance* structure. This leads to a possibility where property subsetting also has a diamond (or even more complicated) structure.

Union properties are the last extension mechanism presented in MOF 2.0 which we will discuss in this paper. If a property is subsetted by other properties, we say that it is a union property. It is not necessary to declare a property as a union, since a designer of a metamodel cannot know in advance if a new subset property will be defined in the future. The UML 2.0 Infrastructure also introduced the concept of derived union. According to page 126 of [16], a derived union property can be seen as the strict union of its subsets. A slot with a property that is a derived union cannot contain elements that do not appear in any of its subsets. Another way to define derived content is to create an arbitrary query operation. This has been done in the Eclipse Modeling Framework using so called *volatile* attributes as explained in [7]. This way, the contents of a slot are defined by evaluating the associated query. The drawback is that there is no strict

mathematical relationship between the derived property and any other properties. The benefit is that it does not restrict the metamodel creator in any way.

2.1 Metamodels

Based on the previous discussion, we can now present a simple metamodeling language that contains the core concepts of MOF and UML 2.0. We describe all metamodels as the tuple $MM = (C, P, \mathit{generalizations}, \mathit{properties}, \mathit{characteristics})$, where C is a set of classes, P a set of properties and $C \cap P = \emptyset$. We define the generalizations of a class with the function $\mathit{generalizations} : C \to \mathcal{P}(C)$. We ignore classes that represent primitive datatypes such as integers, strings and enumeration values without loss of generality. We denote by $\subseteq_c$ the extended generalization between classes that is defined as the reflexive transitive closure of the generalization relation: $\subseteq_c \stackrel{\text{def}}{=} \{(c_1, c_2) \cdot c_2 \in \mathit{generalizations}(c_1)\}^*$. It is a partially ordered set under the assumption that the generalization graph is acyclic.

The properties of a class is given by the function $\mathit{properties} : C \to \mathcal{P}(P)$. Every value of the function *properties* is a disjoint subset of P. Thus, we can define $\mathit{owner} : P \to C$ which denotes the unique owner c of a property p where $p \in \mathit{properties}(c)$. The effective properties of a class are those defined by the class itself and transitively by any of its generalizations.

Finally, the characteristics of a property represent constraints for the elements that can be contained in a slot of that property. We define $\mathit{characteristics} \stackrel{\text{def}}{=} (\mathit{lower}, \mathit{upper}, \mathit{opposite}, \mathit{ordered}, \mathit{composite}, \mathit{derived}, \mathit{supersets})$ as a tuple of functions detailing the properties further. The multiplicity constraints is defined by $\mathit{lower} : P \to \mathbb{Z}^{0+} \setminus \infty$ and $\mathit{upper} : P \to \mathbb{Z}^{+}$. Each property has an opposite property represented by $\mathit{opposite} : P \to P$ that is a bijective function. The opposite of a property cannot be itself but every property is the opposite of its opposite. The function $\mathit{ordered} : P \to \mathbb{B}$ is true if a property is ordered. For example the parameters in an operation should be ordered. The function $\mathit{composite} : P \to \mathbb{B}$ is true if a property is composite. For example, the property that represents the contents of a package is a composite, since a package owns its contents. Finally, there are two characteristics that represent the new property mechanism: $\mathit{derived} : P \to \mathbb{B}$ is true if a property is a derived union while $\mathit{supersets} : P \to \mathcal{P}(P)$ represents the set of properties of which a property is a subset. The graph representing the property superset relation $(P, \{(p1, p2) \cdot p_2 \in \mathit{supersets}(p_1)\})$ must be acyclic.

For convenience, we define the function $\mathit{subsets} : P \to \mathcal{P}(P)$ as the inverse of supersets. We denote subsetting between properties by the $\subseteq_p$ relation, i.e., $\subseteq_p \stackrel{\text{def}}{=} \{(p, q) \cdot q \in \mathit{supersets}(p)\}^*$. We define $a \subset b \stackrel{\text{def}}{=} a \subseteq b \wedge a \neq b$ for both $\subset_c$ and $\subset_p$. Finally, we denote by $s \prec t$ that s is a direct subset of t, i.e., $s \prec t \stackrel{\text{def}}{=} s \subset t \wedge \neg(\exists u \cdot s \subset u \subset t)$. The expression $s \,||\, t$ is defined as $\neg(s \subseteq t) \wedge \neg(t \subseteq s)$, i.e., there is no order defined between s and t.

The notable omission is that we cannot describe nonuniqueness (i.e., bags) with the above definitions. This characteristics exists in UML/MOF but our current formalization cannot cope with it. With some modifications, our framework could understand unordered bags, but ordered bags would still be an issue.

2.2 Models

We define $\mathcal{M} = \{M \cdot M = (E, type, slots, S, property, elements)\}$ as the infinite set of all models in our framework. M comprises all the models in a system at some specific time. E is a finite set of elements and S is a finite set of slots. Each element in E has a type defined by a class in a metamodel, $type : E \to C$, and a set of slots defined by the function $slots : E \to \mathcal{P}(S)$. Every value of the function *slots* is a disjoint subset of S. Thus, we can define $slotowner : S \to E$ which denotes the unique owner e of a slot s where $s \in slots(e)$. Each slot corresponds to a property as defined by the function $property : S \to P$. Slots consist of element references and the function $elements : S \to (E, \prec)$ returns a total ordered set of elements of its argument slot s if $ordered(property(s))$ is true, otherwise $elements : S \to \mathcal{P}(E)$ returns an unordered set of elements. A slot thus describes the connection from its owner element to the elements in the slot. There is no actual ordering defined between the elements in an ordered slot; they merely have an assigned position in it. An element cannot occur twice in a slot.

For convenience, we define the size of a slot to be the amount of elements in that slot: $(\forall s \in S \cdot \ \#s \stackrel{\text{def}}{=} \#elements(s)\ )$. For the elements of an ordered set, we say $s[i]$ to denote the element at the zero-based index i in the ordered set s.

Models are hierarchical structures based on composition properties. We define the function $parent : E \to \mathcal{P}(E)$ to return a set consisting of the parent element of the argument, if any, otherwise the empty set:

$$parent(e) \stackrel{\text{def}}{=} \{x \cdot x \in E \wedge (\exists s \in S \cdot s \in slots(x) \wedge composite(property(s)) \wedge e \in elements(s))\}$$

The slot subsetting relation is $\subseteq_s \stackrel{\text{def}}{=} \{(s,t) \cdot slotowner(s) = slotowner(t) \wedge property(s) \subseteq_p property(t)\}^*$. A slot s (transitively) subsetting another slot t is denoted by $s \subset_s t$.

By definition, if slot s is subsetting slot t, then the contents of s must be a subset of the contents of t. Also, MOF [15] tells us on page 56 that "*The slot's values are a subset of those for each slot it subsets.*" For ordered slots, we also wish to preserve order, i.e., when elements occur in a specific order in s, they should occur in the same order in t, although t might contain more elements in between. We denote $a \prec_x b$ if element a precedes element b in a specific ordered slot x.

There are several constraints that must hold for any models, such as strong typing and at most one parent element for each element. We refer the interested reader to [1] for a more in-depth description of the constraints, but stress three novel constraints due to subsets and unions. The constraints also serve as an invariant which must be maintained by any operation on models.

- The contents of a derived slot is the union of the contents of its subset slots: $(\forall p \in P \cdot derived(p) \Rightarrow (\forall t \in S \cdot property(t) = p \Rightarrow elements(t) \setminus \bigcup\{elements(s) \cdot s \prec t\} = \emptyset))$
- The contents of any unordered slot must also exist in the contents of any superset slots: $(\forall s,t \in S \cdot s \subseteq t \wedge \neg ordered(t) \Rightarrow elements(s) \subseteq elements(t))$
- Similarly to unordered slots, the contents of any ordered slot must also exist in the contents of any superset slots. Additionally, the elements must occur in the same order: $(\forall x,y \in E, s,t \in S \cdot s \subseteq_s t \wedge x \in elements(s) \wedge y \in elements(s) \wedge x \prec_s y \wedge ordered(t) \Rightarrow x \in elements(t) \wedge y \in elements(t) \wedge x \prec_t y)$

These three constraints are specific to derived slots and to unordered and ordered slots with respect to property subsetting. We call them the *inherent subsetting rules*, or ISR.

2.3 Example

Based on the previous definitions, we can describe a part of Figure 2 in a little more detail in Figure 3. We explicitly show slots as filled black circles, and the subsetting relation as a solid line between the circles. We represent a slot visually higher up if it is subsetted by the (connected) slots below it. In the figure, we depict only elements *V1*,*V2* and *E1*. Element *V1* has two slots named *outgoing* and *outgoingRB* such that *outgoingRB* $\subset_s$ *outgoing*. The contents of *outgoingRB* is the set $\{E1\}$. As a consequence of the ISR constraint, the contents of *outgoing* also include *E1*. The slots *from* and *fromRed* are the opposite of outgoing and *outgoingRB* and, as a consequence, they link *E1* to *V1*. In the figure, we see four different partially ordered sets (posets) of slots as dashes ellipses. The first and second poset are isomorphic to each other (as well as the third and fourth) when only considering the slots and the subsetting relation, disregarding the elements they point to. This is always true, since property subsets always come in pairs of two isomorphic posets. Drawing a poset in this way is known as a Hasse diagram [10].

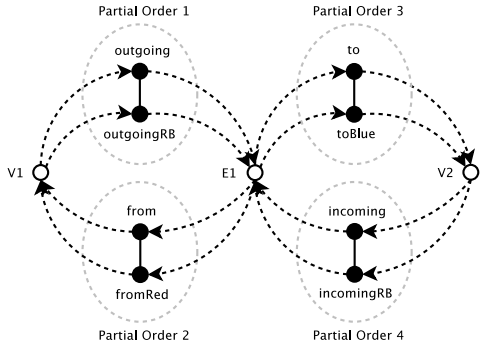

Fig. 3. Part of Figure 2 in More Detail

Let us assume that we want to perform some simple model transformations. The question is what elements should be created and removed from the model and what are the changes to the 8 slots depicted in the figure in order to accomplish these model transformations. We address this problem in the next section.

3 Basic Edit Operations for Models

In this section we present the basic operations to create and delete elements from models as well as to insert to or remove an element from a slot. These four operations are the

basic edit operations for models that are necessary to implement a model repository and a model transformation system.

We should note that a valid model transformation usually involves a sequence of many basic operations. Also, one single basic edit operation can invalidate a slot with respect to the multiplicity constraints. Therefore, we consider a model transformation as a sequence of basic edit operations. As an example, let us assume that we want to create an association *A* between two classes *C1* and *C2* in a model based on a simplified UML with only classes and associations. This requires three basic operations: create *A*, connect *C1* with *A* and connect *C2* with *A*. The association *A* is invalid just after the create operation since an association should connect at least two classes. However, the model should be well-formed after executing all the basic operations.

We define these operations using a pre- and postcondition specification. We first describe element creation and deletion. Then, we describe the case of insertion into ordered or unordered slots and finally the case of removing elements from slots. The pre- and postconditions are described as separate enumerated clauses. All of the clauses in the precondition must hold for the operation to succeed, and all the clauses of the postcondition must be guaranteed by an implementation. For succinctness and understandability of presentation, we only describe the semantics of an operation in the context of one poset. When modifying a slot, similar actions must be taken for the slots in the opposite poset for bidirectionality to hold. This means that the actual operations must, where necessary, be augmented with an additional index parameter for the ordered slots in the opposite poset.

In any pre- or postcondition, the old models are denoted $M = (E, type, slots, S, property, elements)$. In postconditions, the new values of variables are denoted with tick marks. Thus, the new models are denoted $M' = (E', type', slots', S', property', elements')$.

3.1 Element Creation

The operation create : $\mathcal{M} \times C \to \mathcal{M} \times E$ such that $(M', e) = \text{create}(M, c)$ creates a new element of type $c \in C$ and has no preconditions. The new element will also be a root element, i.e., it will not have any parent. The returned value is a tuple of the new models and the new element. The primary postcondition is that there must be exactly one new element in the set of elements. The various model constraints mean that the sets and functions in M must be updated in M' to reflect this change; this leads to more postconditions.

1. $(\exists! e \in E' \cdot E' \setminus \{e\} = E \wedge type'(e) = c)$
2. $type' \cap type = type$
3. $\#S' = \#S + \#\{p \cdot p \in P \wedge (\exists! e \in E' \setminus E \wedge type'(e) \subseteq_c owner(p))\}$
4. $S' \cap S = S$
5. $slots' = slots \cup \{e \to s \cdot e \in E' \setminus E \wedge s \in S' \setminus S\}$
6. $property' = property \cup \{s \to p \cdot s \in S' \setminus S \wedge p \in P \wedge \{\exists! e \in E' \setminus E \cdot type'(e) \subseteq_c owner(p)\}\}$
7. $\#\text{Range}(property' \setminus property) = \#\{p \cdot p \in P \wedge (\exists! e \in E' \setminus E \wedge type'(e) \subseteq_c owner(p))\}$
8. $elements' = elements \cup \{s \to \{\} \cdot s \in S' \setminus S \wedge \neg ordered(property'(s))\}$
   $\cup \{s \to [\,] \cdot s \in S' \setminus S \wedge ordered(property'(s))\}$

The only relevant postcondition is the first one, the rest are implicit or informally understandable from the various model constraints. To avoid too much repetition, we assume that the new values of any variables not mentioned are kept identical to their previous values and that only the necessary changes to fulfill the postconditions are made. We will refrain from listing obvious postconditions and concentrate on the important ones.

3.2 Element Deletion

The operation delete : $\mathcal{M} \times E \rightarrow \mathcal{M}$ deletes an element. We require the element being deleted to have no connections to other elements via its slots. Therefore the precondition for deleting an element e is:

1. $(\forall s \in \mathit{slots}(e) \cdot \ \#s \ = 0)$

The postcondition is that the element must no longer be in the set of elements:

1. $E' = E \setminus \{e\}$

3.3 Element Insertion into an Unordered Slot

Consider an operation $\mathit{insert} : \mathcal{M} \times S \times E \rightarrow \mathcal{M}$ such that $\mathit{insert}(M,s,e)$ inserts element e into slot s. The intuition behind the insertion operation is that all supersets of s must contain the new element e for the ISR constraints to hold. The clauses for the precondition for element insertion into an unordered slot are thus:

1. $\neg \mathit{derived}(\mathit{property}(s))$
2. $\neg \mathit{ordered}(\mathit{property}(s))$
3. $e \notin \mathit{elements}(s)$.
4. $\mathit{type}(e) \subseteq \mathit{owner}(\mathit{opposite}(\mathit{property}(s)))$
5. $(\exists t \in S \cdot s \subseteq_s t \wedge \mathit{composite}(\mathit{property}(t)) \Rightarrow \mathit{parent}(e) \setminus \{\mathit{slotowner}(t)\} = \emptyset$

The clauses state that (1) we are not modifying a derived read-only slot, (2) the slot is unordered, (3) the element must not yet exist in the slot, (4) that we obey the rules of strong typing and (5) we do not create a connection to a second parent for e.

The postcondition for element insertion is simple. We wish element e to be found in the slot s and all its transitive supersets. All the model constraints except for the multiplicity constraints must also hold as a postcondition.

1. $(\forall t \in S \cdot s \subseteq_s t \Rightarrow \mathit{elements}'(t) = \mathit{elements}(t) \cup \{e\})$ (Note $s \subseteq_s s$)

An example of element insertion into an unordered slot can be seen in Figure 4. Again, the Hasse diagram notation means that $t \subset_s q \subset_s p \wedge q \subset_s r \wedge t \subset_s s \subset_s r$. In case (1) of the figure, we have a poset of unordered slots. Suppose we insert an element c into slot q. This requires an insertion of c into slots p and r as well, to maintain the ISR, with the end result shown in case (2). After this, inserting c into slot t also inserts it into slot s, again to maintain the ISR, resulting in case (3). Slots p, q and r are not modified because c already existed in those slots.

It can be noted that in our semantics, an insertion into a slot never modifies any subset of that slot.

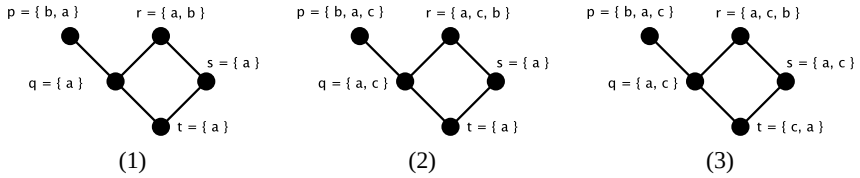

Fig. 4. Example of Inserting an Element into Unordered Slots

3.4 Element Insertion into an Ordered Slot

Subsetting with ordered slots is more complicated than with unordered slots, due to the need to maintain an order between the elements in different slots. We define the operation $insert : \mathcal{M} \times S \times E \times \mathbb{Z}^{0+} \rightarrow \mathcal{M}$ such that $insert(M,s,e,i)$ inserts an element e into a slot s at index i.

We assume there is a function $index : E \times S \rightarrow \mathbb{Z}^{0+}$ which returns the zero-based index of an element in the contents of an ordered slot. A function $lower\_index : \mathbb{Z}^{0+} \times S \times S \rightarrow \mathbb{Z}^{0+}$ is such that $lower\_index(i,x,y)$ returns the index in x where $y[i]$ should be inserted to maintain the subset $x \subseteq_s y$. It is shown in Figure 5 and is used to calculate which restrictions from supersets apply to subsets when inserting an element. As an example, consider what the restriction given by element c (at index position 2) in the superset $[a,b,c,d]$ is to its subset $[a,d]$. Then $lower\_index(2,[a,d],[a,b,c,d])$ returns 1 since c should be inserted between a and d.

```
lower_index(i,s,t) :=
  if t[i] ∈ s then return index(t[i],s)
  do
    if t[i] ∈ s then return index(t[i],s) + 1
     else if i = 0 then return 0
     else i := i − 1
  od
```

```
lift_interval(s,t,[v..w]) :=
  if v > 0 then v' := index(s[v − 1],t) + 1
     else v' := 0
  if w = #s then w' := #t
     else w' := index(s[w],t)
  return [v'..w']
```

Fig. 5. (Left) The *lower_index* Function . (Right) The *lift_interval* Function

A function $lift\_interval : S \times S \times R \rightarrow R$, where R denotes integer intervals is such that $lift\_interval(s,t,[v..w])$ "lifts" the interval $[v..w]$ from s as superimposed on t (when $s \subseteq_s t$). It is shown in Figure 5 and is used to calculate which restrictions from subsets apply to supersets and works as the dual of *lower_index*. As an example, consider the ordered sets $s = [c]$ and $t = [b,c]$. If we were to insert element a at index 0 in s, the corresponding interval for s would be $[0..0]$. This interval is superimposed onto t as the interval $[0..1]$, meaning that the same element can be inserted either before or after b in t without violating the ISR. Thus, $lift\_interval(s,t,[0..0]) = [0..1]$.

The function $indices\_ok : \mathcal{P}(S) \times (S \rightarrow R) \rightarrow \mathbb{B}$ returns true if when executing $indices\_ok(T,F)$ there is a possible way to insert an element into every slot in T such

that the constraints in F are satisfied. Here, $F : S \to R$ is a map from slots to integer intervals $[v..w]$ such that $v \leq w$ where e can be inserted. The function is shown in Figure 6. Here, $\mathrm{Dom}(F)$ returns the domain of function F. Using the *lift_interval* and *lower_index* functions we restrict the possible intervals where e can be inserted into the slots.

$$
\begin{array}{l}
\mathit{indices\_ok}(\emptyset, F) := (\forall t \in \mathrm{Dom}(F) \cdot F(t) \neq \emptyset) \\
\\
\mathit{indices\_ok}(T, F) := \\
\quad (\exists t \in T \cdot (\forall u \in T \cdot t \not\supset u) \\
\qquad \wedge R \stackrel{\text{def}}{=} \cap\{\mathit{lift\_interval}(c, t, [v..w]) \cdot (\forall c \cdot s \subseteq_s c \prec t \wedge F(c) = [v..w])\} \\
\qquad \Rightarrow \mathit{indices\_ok}(T \setminus \{t\}, F[t \mapsto R \cap F(t)]))
\end{array}
$$

Fig. 6. The *indices_ok* Function

The precondition of inserting into an ordered slot is otherwise identical to the case when inserting into an unordered slot, except for the check for an ordered slot and that there exists an extra clause which calculates if the insertion into the slot and its transitive supersets is at all possible without violating the ISR.

1. $\neg \mathit{derived}(\mathit{property}(s))$
2. $\mathit{ordered}(\mathit{property}(s))$
3. $e \notin \mathit{elements}(s)$
4. $\mathit{type}(e) \subseteq_c \mathit{owner}(\mathit{opposite}(\mathit{property}(s)))$
5. $(\exists t \in S \cdot s \subseteq_s t \wedge \mathit{composite}(\mathit{property}(t)) \Rightarrow \mathit{parent}(e) \setminus \{\mathit{slotowner}(t)\} = \emptyset$
6. $\mathit{indices\_ok}(\{t \cdot s \subset_s t\},$
   $\{s \mapsto [i..i]\}$
   $\cup \{t \mapsto [\mathit{lower\_index}(\mathit{index}(e,u), t, u)..\mathit{lower\_index}(\mathit{index}(e,u), t, u)] \cdot s \subset_s t \wedge t \subseteq_s u \wedge e \in \mathit{elements}(u)\}$
   $\cup \{t \mapsto [0, \#t\ ] \cdot s \subset_s t \wedge \neg(\exists u \cdot t \subseteq_s u \wedge e \in \mathit{elements}(u))\}$
   $)$

The intuition behind the last clause in the precondition and the definition of the *indices_ok* function is that we calculate the range restrictions of e which exist in any super- or subsets onto the other slots. The F function is initially created by describing constraints from supersets. F is created from three different clauses. The first, $s \mapsto [i..i]$, constrains e to be inserted at exactly index i. The second does similarly for supersets which have a superset that already has e, whereas the third initially allows all indices to be candidates for insertion. This initialization makes sure that F is restricted by the the elements e that already exist in any supersets of s. Note that any slot o such that $o \subset_s t \wedge s \subset_s t \wedge o \,||\, s$ is outside of the transitive superset closure of s and any restrictions from it will already be visible in t and thus it is not necessary to include o in F.

Then, *indices_ok* calculates the constraints from subsets and does set intersection to calculate whether an insertion is possible. The actual function takes all supersets T and picks one $t \in T$ which is a bottom element, which must exist since the slots in T are part

of a finite poset. It then imposes all intervals from subset slots c (such that $s \subseteq_s c \prec t$) onto t, also including the initial constraint on t. It then recurses with a modified F until T is empty. The notation for a modified function is $f[x \mapsto y]$ which returns a new function f' such that $(\forall z \neq x \cdot f'(z) = f(z))$ and $f'(x) = y$.

We claim, without proof, that if the final mapping F contains only nonempty intervals, it is possible to successfully insert e into s at index i. The postcondition is:

1. $elements'(s)[i] = e$
2. $(\forall t \in S \cdot s \subseteq_s t \wedge e \notin elements(t) \Rightarrow elements'(t) \setminus \{e\} = elements(t)$
   $\wedge e \in elements'(t))$

The current definitions do not tell us the exact index where to insert e into any superslot of s, only that a combination of indices exists; an index i_t for a superslot t of s must exist somewhere in the range given by $F(t)$.

An example of element insertion can be seen in Figure 7. Case (1) is the initial configuration of the slots w, x, y and z. Let us assume an insertion of element c into slot w at index position 0 occurs. The returned slot ranges where c should be inserted raises the possibilities in cases (2) to (5), depending on whether c is inserted onto the left or right side of either a in slot y or b in slot z. Cases (2), (3) and (4) are correct solutions and our postcondition does not prefer any particular one over the another. Case (5) is not legal, because slot x cannot maintain the superset relationship as enforced by both slots y and z, as element c should occur both before a and after b in the ordered set. It is up to the implementation to choose one of the correct solutions, perhaps with guidance from the user.

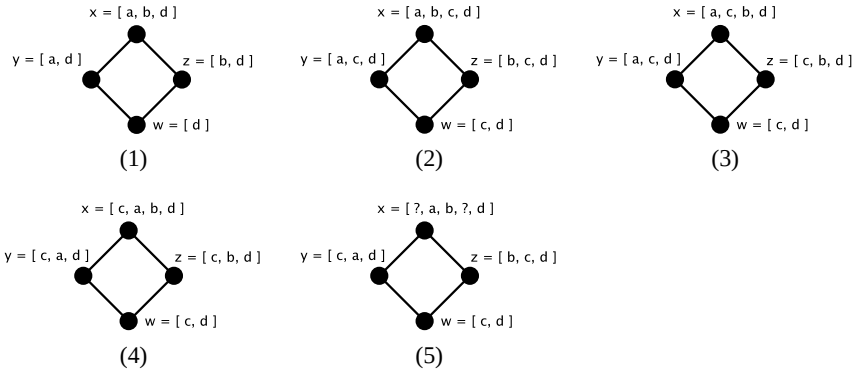

Fig. 7. Example of Inserting an Element into Ordered Slots

3.5 Element Removal from a Slot

The operation $remove : \mathcal{M} \times S \times E \rightarrow \mathcal{M}$ is defined such that $remove(M, s, e)$ removes the element e from s and all its subsets, as well as from those supersets which would not

acquire e via some other subset which is not comparable to s. Element removal from an ordered slot is identical to element removal from an unordered slot since removing a specific element from an ordered slot does not alter the relative position of the other elements in the slot.

The precondition requires that a derived slot is not being modified and that the element must exist in the slot:

1. $\neg derived(property(s))$
2. $e \in elements(s)$

The postcondition:

1. $(\forall r \in S \cdot r \subseteq_s s \Rightarrow elements(r) = elements'(r) \cup \{e\} \wedge e \notin elements'(r))$
2. $(\forall t \in S \cdot s \subset_s t \wedge \neg(\exists m \in S \cdot m \subset_s t \wedge m \,||\, s \wedge e \in elements(m))$
   $\Rightarrow elements(t) = elements'(t) \cup \{e\} \wedge e \notin elements'(t))$

Both clauses in the postcondition are interesting. The first clause states that a removal from a slot triggers a removal from any subset, so that the ISR can hold. This can be contrasted with the insertion operation, which does not modify any subsets. The second clause states that a removal from a slot triggers a conditional removal from any superset. An interesting feature of the clause is shown in Figure 8. If we have an initial setting as in case (1) and remove a from z, the clause requires that a is removed from x as shown in case (2), although this is not necessary to maintain model consistency. However, we believe that this feature is the intended usage by the modeling standards. Inserting into a subset triggers insertion in all supersets, and so dually a removal from a subset ought to trigger a removal from all supersets. A similar chain of reasoning has been reported by Markus Scheidgen [17].

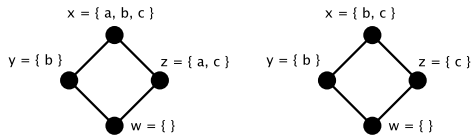

Fig. 8. Removing a from an Unordered Slot z

As an example where the second clause is necessary, consider Figure 9 with the initial setting as in case (1). Assume we wish to remove a from y. An incorrect approach is the removal of a from supersets and subsets, which would leave x without a, but z with a intact, violating the ISR, as shown in case (2). A correct option would be to remove a also from z, as shown in case (3), but our opinion is that this "snowball effect" of removing a reduces the usefulness of subsets; slot y should affect slot z as little as possible, since they are not comparable in the Hasse diagram. Our postcondition ensures that a must be removed from w and y, but not from x, because z still contains a; this is seen in case (4).

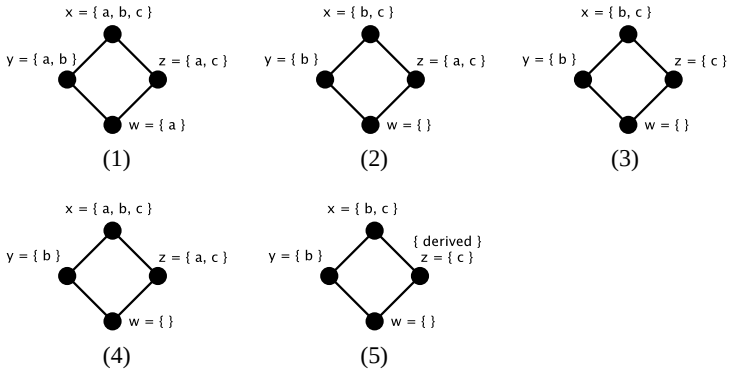

Fig. 9. Different Scenarios for Removing *a* from an Unordered Slot *y*

Another interesting case is the ISR rule for derived slots. If (and only if) *z* is marked as derived, we must remember that its elements must be found in the union of its subsets. In case (5), *a* is removed from *y* which leads to it being removed from *w* as well. As *z* is marked as derived, *a* must also be removed from it, since *z* does not have any other subset containing *a*. This in turn leads to *a* being removed from *x*!

3.6 Implementation of Edit Operations in a Modeling Toolkit

We do not discuss the actual implementation of the basic edit operations in this article due to space restrictions. However, we have implemented the metamodeling language with the operations as described in this article in our modeling tool Coral, with details defined in [1]. We have tested the implementation extensively and found no consistency errors or omissions. Coral is open source and available at `http://mde.abo.fi/`.

We know of no other tools that support subsets as extensively as proposed in this article, even with different semantics. At the time of writing, the Eclipse EMF model repository does not implement subsets, although the feature is being planned.

4 Related Work

Several others have studied the formalization of the metamodel and model layers in the past, for example [5, 3, 9]. Our contribution comes from the definitions of property subsets, which neither metamodeling nor traditional object oriented language descriptions explain.

Several authors use association inheritance without defining exact semantics, and some say that it denotes covariance. An example of this covariant specialization [8] is the multilevel metamodeling technique called VPM by Varro and Pataricza [18], which also limits itself to single inheritance. We argue that property subsetting is not the same concept as covariant specialization, and requires different semantics.

Carsten Amelunxen, Tobias Rötschke and Andy Schürr are authors to the MOFLON tool [4] inside the Fujaba framework [12]. MOFLON claims to support subsetting, but no description of the formal semantics being used is included. It is not clear if their tool works in the context of subsets between ordered slots, or with diamond inheritance with subsetting.

Markus Scheidgen presents an interesting discussion of the semantics of subsets in the context of creating an implementation of MOF 2.0 in [17]. To our knowledge, this has been so far the most thorough attempt to formalize subset properties. The approach is slightly different in that a slot modification creates an *update graph* of slots, so that a later modification at some other slot in the update graph actually updates all the associated slots. The actual operational semantics are unfortunately not described in detail. In comparison, we do not have to create or maintain any update graphs. Furthermore, our contribution not only discusses but also defines pre- and postconditions and implementations for the operations for ordered and unordered sets. It is also not clear if the work by Scheidgen supports diamond subsets or ordered sets, both of which are used in the UML 2.0.

The object-oriented and database research communities are also researching a similar topic, although it is called relationship or association inheritance, or first-class relationships. In [6], Bierman and Wren present a simplified Java language with first-class relationships. In contrast with our work, they do not support multiple inheritance, bidirectionality or ordered properties; all of these constructs are common in modeling and in the UML 2.0 specification. However, relationship links are explicitly represented as instances, and they can have additional data fields (just like the AssociationClass of UML). As the authors have noticed, the semantics of link insertion and deletion is not without problems. Albano, Ghelli and Orsini present in [2] a relationship mechanism for a strongly-typed object-oriented database programming language. It also handles links as relationship instances, but without additional data fields. Multiple inheritance is supported, but ordered slot contents are not.

5 Conclusions

MOF 2.0 provides new property characteristics: subsets, (derived) unions and redefinitions. However, it does not describe these concepts in detail, not even informally, and therefore they cannot be applied in practice. In this article, we have first described a simple formalization of metamodels and models and then presented pre- and postconditions for basic operations on element creation and deletion and slot modification, taking into account subsets and derived unions. It must be stressed that we do not cover several important aspects of MOF 2.0, such as association end ownership or navigability. They are not in the scope of this article.

We consider that the definition of these concepts is not as straightforward as one may think and it requires an extensive study. There is an imminent need in the modeling community to standardize on one formalization of subsets and derived unions, so that tools implementing MOF 2.0 and UML 2.0 can be interoperable. The semantics described in this article are one proposal and we hope it spurs further interest and discussion. We have avoided using OCL or any other modeling standard in order to be able

to present a relatively small and self-contained description of the core of these OMG standards with respect to subsetting. Furthermore, the idea of subsetting is intriguing, since it is a new construct for modeling relationships between classes and objects, and thereby brings a novel idea to the software modeling and object-oriented community.

The authors would like to thank Patrick Sibelius for insightful discussions. Marcus Alanen would like to acknowledge the financial support of the Nokia Foundation.

References

1. Marcus Alanen and Ivan Porres. Subset and union properties in modeling languages. Technical Report 731, TUCS, Dec 2005.
2. Antonio Albano, Giorgio Ghelli, and Renzo Orsini. A Relationship Mechanism for a Strongly Typed Object-Oriented Database Programming Language. In *Proceedings of the 17th Conference on Very Large Databases, Morgan Kaufman pubs. (Los Altos CA), Barcelona*, 1991.
3. José Álvarez, Andy Evans, and Paul Sammut. MML and the Metamodel Architecture. In Jon Whittle, editor, *WTUML: Workshop on Transformation in UML 2001*, April 2001.
4. Carsten Amelunxen, Tobias Rötschke, and Andy Schürr. Graph Transformations with MOF 2.0. In Holger Giese and Albert Zündorf, editors, *Fujaba Days 2005*, September 2005.
5. Thomas Baar. Metamodels without Metacircularities. *L'Objet*, 9(4):95–114, 2003.
6. Gavin Bierman and Alisdair Wren. First-class relationships in an object-oriented language. In *Workshop on Foundations of Object-Oriented Languages (FOOL 2005)*, January 2005.
7. Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, August 2003.
8. Giuseppe Castagna. Covariance and Contravariance: Conflict without a Cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, May 1995.
9. Tony Clark, Andy Evans, and Stuart Kent. The Metamodelling Language Calculus: Foundation Semantics for UML. In H. Hussmann, editor, *Fundamental Approaches to Software Engineering. 4th International Conference, FASE 2001*, volume 2029 of *LNCS*, pages 17–31, 2001.
10. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002.
11. Anneke Kleppe, April 2003. Discussion on the mailing-list puml-list@cs.york.ac.uk.
12. Ulrich A. Nickel, Jörg Niere, and Albert Zündorf. Tool demonstration: The FUJABA environment. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE)*, pages 742–745. ACM Press, 2000.
13. OMG. MOF 2.0 Query / View / Transformation Final Adopted Specification. OMG Document ptc/05-11-01, available at www.omg.org, 2005.
14. OMG. UML 2.0 Superstructure Specification, August 2005. Document formal/05-07-04. Available at `http://www.omg.org/`.
15. OMG. Meta Object Facility (MOF) Core Specification, version 2.0, January 2006. Document formal/06-01-01, available at `http://www.omg.org/`.
16. OMG. UML 2.0 Infrastructure Specification, March 2006. Document formal/05-07-05, available at `http://www.omg.org/`.
17. Markus Scheidgen. On Implementing MOF 2.0—New Features for Modelling Language Abstractions. July 2005. Available at `http://www.informatik.hu-berlin.de/~scheidge/`.
18. Dániel Varró and András Pataricza. VPM: A visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. *Journal of Software and Systems Modeling*, 2(3):187–210, October 2003.