Ontological Meta-Properties of Derived Object Types

Giancarlo Guizzardi

Ontology and Conceptual Modeling Research Group (NEMO)
Computer Science Department,
Federal University of Espírito Santo (UFES), Brazil
gguizzardi@inf.ufes.br

Abstract. In this paper, we revisit a number of classical formal meta-properties that have been used in the conceptual modeling and ontology engineering literature to provide finer-grained distinctions among the category of Object Types. These distinctions constitute an essential part of relevant existing approaches, in particular, the ontology-driven conceptual modeling language OntoUML, and the ontology and taxonomy evaluation methodology OntoClean. The idea in this paper is to investigate the interaction between these meta-properties and Derived Object Types, i.e., Object Types which extensions are dynamically inferred via Derivation Rules. The contributions here are two-fold: firstly, we revisit two classical Derivation Patterns and prove a number of results that can be used to infer the modal meta-properties of Derived Types from those of the types participating in the associated derivation rules; secondly, we demonstrate how these results can be applied in the automated support for model construction in OntoUML.

Keywords: Ontological Foundations, Meta-Properties, Derived Object Types

1 Introduction

In recent years, there has been a growing interest in the use of Ontologically Well-Founded Conceptual Modeling languages to support the domain analysis phase in Information Systems Engineering. OntoUML is an example of a conceptual modeling language whose metamodel has been designed to comply with the ontological distinctions and axiomatic theories put forth by a theoretically well-grounded Foundational Ontology [1]. This language has been successfully employed in a number of industrial projects in several different domains, ranging from Petroleum and Gas [2] to News Information Management [3]. In fact, recently, it has been considered as a possible candidate for contributing to the OMG SIMF (Semantic Information Model Federation) standardization *request for proposal* [4] after a significant number of successful applications in real-world engineering settings [5,6].

One of the striking features of OntoUML is the fact that its modeling primitives are defined in terms of ontological meta-properties that not only give a precise semantics to the language´s primitives but also serve as a methodological guideline for helping modelers to choose the most suitable category to model elements from the universe of discourse [7]. OntoUML addresses all the classical primitives used in structural conceptual modeling, including Classes, Relations, Weak Entities, Attributes, Attribute

Value Spaces (Datatypes) and different types of Part-Whole Relations. However, in this paper we will focus only on Object Types and their taxonomic relations, which are certainly among the most fundamental constructs in structural conceptual models.

In [1], we have shown that traditional conceptual modeling and knowledge representation languages (e.g., ER, UML, OWL) present a clear case of ontological incompleteness w.r.t. the representation of Object Types. In these languages, there is one single concept of object type and, hence, one single type of classification relation. In contrast, as demonstrated in [1,7], from an ontological and cognitive point of view, there are number of different categories of object types implying different types of classification relations among object types and their instances. As discussed in depth in [1,7], these categories provide a formally characterized, ontologically grounded and empirically supported version of modeling primitives which have been used and defined in an *ad hoc* manner in the history of conceptual modeling (e.g., Kind, Role, Mixin, Phase or State, etc.).

Among the most important meta-properties used to define the aforementioned distinctions, we have the meta-properties which have a modal nature, in particular, the so-called meta-properties of *Rigidity* and *Non-Rigidity* (including *Semi-Rigidity* and *Anti-Rigidity*). These meta-properties have also been used in the methodological guidelines of one of the most important and successfully employed Ontology Evaluation Methodologies, namely OntoClean [8,9] and have also been incorporated into ORM 2.0 [10].

In recent papers, we have demonstrated that the formal nature of these meta-properties, and the manner in which they constrain how the OntoUML modeling primitives can be combined, account for an important mechanism for building automated tools for formal verification [11] and conceptual model validation via visual simulation [12]. Moreover, as shown in [13], the constrained manner in which these primitives can be combined makes them converge to a number of *Ontological Design Patterns* and, hence, OntoUML can be characterized as a Pattern Language. In the same article, we present a computational tool that takes advantage of this characteristic of the language to help modelers to build ontologically sound conceptual models by assembling these models via the chained combination of these patterns.

However, up to this point, we have only analyzed the so-called *Base Types*, i.e., types which are explicitly defined in a conceptual model and which instance populations have to be explicitly asserted. In contrast, *Derived Types* are types which are derived from other (Base and Derived) Types via derivation rules. Again, in contrast to Base Types, the populations of Derived Types are dynamically inferred via their associated derivation rules. In [14], Olive discusses in depth the importance of derived types for conceptual modeling and presents a number of patterns for derived types with associated formal derivation rules.

In this paper, we take a step further in developing theoretical results for the foundational of conceptual modeling and ontology engineering, in general, and OntoUML and OntoClean (but also to some extent ORM), in particular. Firstly, we analyze how these modal meta-properties of Object Types interact with patterns for Derived types as defined in [14]. Moreover, we show how the modal properties of derived types can be inferred from those of the other types participating in the associated derivation rules (section 3). In particular, in the case of OntoUML, we manage to show how the particular stereotypes (representing ontological categories) of these derived types can be inferred from those of the types participating in the derivation rules (section 4).

In section 2, we first present the background information for this paper. Firstly, we present two of Olive´s patterns for Derived Object types. Moreover, we present the modal meta-properties of Object Types (Rigidity, Non-Rigidity, Anti-Rigidity and Semi-Rigidity) underlying the theories of OntoUML and OntoClean, and briefly present the modeling primitives of OntoUML which are generated from the variations of these meta-properties.

Section 5 elaborates on final considerations and directions for future work.

2 Background: Derived Types, Ontological Modal Meta-Properties and Object Type distinctions in OntoUML

2.1 Derived Types and Derivation Patterns

Olive [14] presents and formally characterizes some properties of a number of categories of Derived Object types with their associated derivation rules. These include *Derivation by Participation*, *Derivation by Intersection*, *Derivation by Union* and *Derivation by Exclusion*. These different patterns are associated with different types of derivation rules which imply different types of constraints. Here, due to space limitations, we focus on *Derivation by Union* and *Derivation by Exclusion*.

Derivation by Union: The pattern of Derivation by Union takes place when we have a type T whose extension is necessarily defined as a union of the extension of a number of other types $T_1 \ldots T_n$. As shown in [14], a Derivation by Union constraint implies a number of specialization relations between $T_1 \ldots T_n$ and the derived common supertype T (symbolized as /T, following UML), such that $T_1 \ldots T_n$ and T form together a complete *generalization set (GS).* For instance, in figure 1.b, the type Student is defined by union of types Graduate Student and Undergraduate Student. In other words, all instances of Student are instances of either Graduate Student or Undergraduate Student. Formally, let W be a set of possible worlds and let ext(T,w) be function that maps a generic type T to its extension in world $w \in W$. In this case can state that: for all $w \in W$, we have that ext(Student,w) = ext(UndergraduateStudent,w) $\cup$ ext(GraduateStudent,w). In general, we can present the following formal definition:

Definition 1 (Derivation by Union): if the type T is derived by union from types $T_1 \ldots T_n$ such that T is a common supertype of $T_1 \ldots T_n$ forming a generalization set then for all worlds $w \in W$, we have that $ext(T,w) = ext(T_1,w) \cup \ldots \cup ext(T_n,w)$. ■

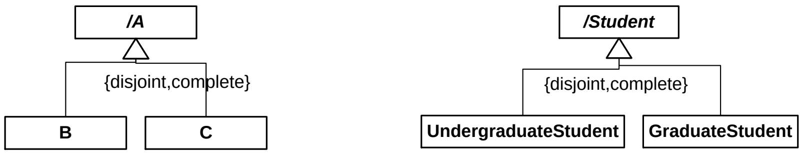

Fig.1. Pattern for *Derivation by Union* (a-left) and its exemplification (b)

The generalization set formed in the *derivation by union* pattern must be complete. However, in general, the types $T_1 \ldots T_n$ with a common supertype T do not have to be disjoint. However, as discussed by many authors in areas ranging from Conceptual modeling, Logic and Philosophy [15], defining generalization sets which are both disjoint and complete (i.e., partitions) of a common supertype is a design principle which should be pursued in taxonomic structures accounting for ontological, logical and methodological advantages. For this reason, when referring to the *derivation by union* pattern in this article, we mean a *derivation by disjoint union* pattern.

Derivation by Exclusion: The pattern of Derivation by Exclusion takes place when we have a type T_x whose extension is necessarily defined as the complement of the extension a number of other types $T_1 \ldots T_n$ w.r.t. a type T. As shown in [14], a Derivation by Exclusion constraint implies a number of specialization relations between $T_1 \ldots T_x \ldots T_n$ and the common supertype T, forming a *disjoint* and *complete generalization set.* For instance, in figure 2.b, the type Deceased Person is defined by exclusion as the complement of type Living Person w.r.t. the type Person. In other words, all instances of Deceased Person are instances of Person which are not Living Persons. Formally, in this case can state that: for all $w \in W$, we have that ext(DeceasedPerson,w) = ext(Person,w) - ext(LivingPerson,w). In general, we define the following constraint for this type of Derivation.

Definition 2 (Derivation by Exclusion): if the type T_x is derived by exclusion from types $T_1 \ldots T_n$ w.r.t. their common supertype T forming a Generalization Set then for all worlds $w \in W$, we have that $\text{ext}(T_x,w) = \text{ext}(T,w) - (\text{ext}(T_1,w) \ldots \cup \ldots \cup \text{ext}(T_n,w))$. ■

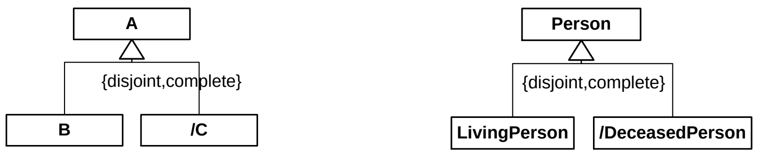

Fig.2. Pattern for *Derivation by Exclusion* (a-left) and its exemplification (b)

2.2 The Ontological Meta-Properties among the Categories of Object Types and Associations

OntoUML has been proposed as an extension of UML that incorporates in the UML 2.0 original metamodel a number of ontological distinctions and axioms put forth by the Unified Foundation Ontology (UFO) [1]. In this work, we focus our discussion on a small fragment of this metamodel. This fragment discusses an extension of the *Class* meta-construct in UML to capture a number of ontological distinctions among the Object Types categories proposed in UFO. Besides incorporating modeling primitives that represent these ontological distinctions, the extended OntoUML metamodel includes a number of logical constraints that govern how these primitives can be combined to form consistent conceptual models. In what follows, we briefly elaborate on the aforementioned distinctions and their related constraints for the fragment discussed in this article. For a fuller discussion and formal characterization of the language, the reader is referred to [1].

A fundamental distinction in OntoUML among the category of object types is the one between types which provide a uniform principle of identity, individuation and counting for their instances - termed ***Sortal*** types, and types which merely define characterizing properties for instances of multiple ***Sortal*** types – termed ***Characterizing*** or ***Dispersive*** types. For instance, contrast the Sortal types *Person* and *Car*, with the characterizing types *Insured Item*, *Colored Object* or *Spatially Extended Object*. Whilst the first categories of types provide a uniform principle of identity for their instances, the latter category of types merely define properties which are common to instances of different sortal types. For example, the type *Spatially Extended Object* describes properties which are common to all its instances but it cannot provide a uniform principle of identity for them. In other words, different instances of Spatially Extended Object (e.g., Cars, Houses, Persons, Watches, etc.) will obey different principles of identity which are provided by the respective sortal types they instantiate. As discussed in depth in [7], this distinction is one of the most supported ones in Philosophy of Language and there is a significant amount of empirical evidence in Cognitive Psychology supporting the claim that it is a fundamental cognitive distinction.

Orthogonally to this basis distinction, we can make another distinction among Object Types by considering a formal modal meta-property named *Rigidity* (henceforth symbolized R+) [7,9]. In short, a type T is rigid iff for every instance x of that type, x is necessarily an instance of that type. Formally, we have that:

Definition 3 (Rigidity R+): a type T is rigid if every instance of that type is necessarily (in a modal sense) an instance of that type. In other words, the extension of type T is world invariant, i.e., instances of T in any world w are its instances in all possible worlds. Hence, for all w,w' $\in$ W we have that ext(T,w) = ext(T,w'). ■

In contrast, a type T' is anti-rigid (henceforth symbolized R~) iff for every instance y of T', there is always a possible world in which y is not an instance of T'. In other words, it is possible for every instance y of T' to cease to be so without ceasing to exist (without losing its identity) [7]. Formally,

Definition 4 (Anti-Rigidity R~): a type T is anti-rigid if every instance of that type can possibly (in a modal sense) cease to be an instance of that type. In other words, for all worlds w $\in$ W and for all instances x such that x $\in$ ext(T,w) we have that there is a world w' $\in$ W such that x $\notin$ ext(T,w'). ■

A stereotypical example of this distinction can be found by contrasting the rigid type Person with the anti-rigid type Student. Whilst every instance of Person cannot cease to be an instance of Person without ceasing to exist, instances of Student can move in and out the extension of that class and still exist preserving its identity, i.e., being the same Person [7]. One should notice, however, that Anti-Rigidty is not the logical negation of Rigidity, it is stronger than that. The logical negation of Rigidity is termed Non-Rigidity (henceforth symbolized R-) and is defined as follows:

Definition 5 (Non-Rigidity R-): a type T is non-rigid if there is an instance of that type which can possibly (in a modal sense) cease to be an instance of that type. In other words, there is a world w ∈ W and an instance x such that x ∈ ext(T,w) such that for another world w' ∈ W we have that x ∉ ext(T,w'). ■

From the above definition, it is clear that Anti-Rigidity implies Non-Rigidity, i.e., Anti-Rigidity is a strong case of Non-Rigidity in which the contingent classification in a type holds for all instances of that type T. A complement of Anti-Rigidity w.r.t. Non-Rigidity is named Semi-Rigidity (henceforth symbolized R¬), which is a meta-properties much more common than Non-Rigidity in conceptual Models. Semi-Rigidity of type T means that the classification in T is necessary for some of its instances and contingent for others. In other words, *T is rigid for some of its subtypes and anti-rigid for others [7,9]*. Formally, we have that:

Definition 6 (Semi-Rigidity R¬): a type T is semi-rigid if it is Non-Rigid but not Anti-Rigid. ■

Suppose a conceptualization in which People and Houses can optionally have an Insurance Policy but Cars are obliged to have one. In other words, instances of Person and House are contingently *Insured Items* but Cars are necessarily Insured Items. In this case, Insured Item will be an example of a Semi-Rigid type.

Sortal Rigid types can be related in a chain of taxonomic relations. For instance, the rigid types Man and Person are related such that the former is a specialization of the latter. The type in the root of a chain of specializations among sortal rigid types is termed a Kind (e.g., Person) and the remaining rigid types in this chain are named Subkinds (e.g., Man, Woman). As formally demonstrated in [1], we have the following constraints involving Kinds and Subkinds: *(i) every object in a conceptual model must be an instance of exactly one kind, which means that kinds cannot specialize other kinds; (ii) as consequence, we have that for every object type T in OntoUML, T is either a kind or it is the specialization of exactly one ultimate kind. In other words, all sortals in a conceptual model which are not kinds must specialize one unique kind*; *(iii) by definition, subkinds cannot specialize kinds.*

Among the anti-rigid Sortal types, we have again two subcatetories: Phases and Roles. In both cases, we have cases of dynamic classification, i.e., the instances can move in and out of the extension of these types without any effect on their identity. However, while in the case of Phase these changes occur due to a change in the intrinsic properties of these instances, in the cases of Role, they occur due to a change in their relational properties. We contrast the types Child, Adolescent, Adult as phases of a Person with the Roles Student, Husband or Wife. In the former case, it is a change in the intrinsic property *age* of Person which causes an instance to move in and out of the extension of these phases. In contrast, a Student is a role that a Person plays when related to an Education Institution and it is the establishment (or termination) of this relation that alters the instantiation relation between an instance of Person and the type Student. Analogously, a Husband is a role played by a Person when married to (a Person playing the role of) Wife. *As formally proved in [7], rigid types cannot specialize (cannot be subtypes of) anti-rigid types.*

Sortal types are always Rigid or Anti-Rigid. In contrast, among the categories of Characterizing types, we have both Rigid, Anti-Rigid and Semi-Rigid types. Rigid Characterizing types describe essential (i.e., modally necessary) properties which are common to instances of different kinds. These are named *Category*. An example of a Category is Physical Object. Every Physical Object is necessarily a Physical Object. However, there is no uniform principle of identity obeyed by all types of Physical Objects (e.g., Cars, Persons, Houses, Oranges). An example of an Anti-Rigid Characterizing type is Customer: every customer can cease to be a Customer. However, different instances of Customer can obey different principles of identity belonging to different kinds (e.g., People and Organizations). These sorts of characterizing types are named *RoleMixins* in OntoUML. Finally, we have already encountered an example of a Semi-Rigid Characterizing type in this article, namely, Insured Item. On one hand, it is a characterizing type classifying instances of different kinds. On the other hand, it is necessary for some of its instances (e.g., the instances of the subtype Car) and contingent to others (e.g., the instances of the subtype Person). These characterizing types are termed *Mixin* in OntoUML. *As also formally proved in [7], characterizing types cannot specialize (be subtypes) or sortals*. This is due to the fact that principles of identity are inherited in a specialization chain. Thus, if a characterizing type (e.g., Red Object) specializes a Sortal (e.g., Apple), it inherits the principle of identity of latter and becomes a Sortal (e.g., Red Apple).

3. Deriving Modal Meta-Properties of Derived Object Types

In this section, we demonstrate how the modal meta-properties of derived types for two of the Derivation Patterns proposed in [14] can be inferred from the modal meta-properties of the types participating in the derivation. Firstly, we refrain from using directly the ontological categories of OntoUML so that the results can be made more general, i.e, can be re-used in other conceptual modeling and ontology engineering methodologies that make use of the same notions such as, for instance, OntoClean.

3.1 Derivation by Union

Let us start with the case of *derivation by union* (exemplified in the pattern of figure 1.a). Let us first suppose that B in that figure is a rigid type. In the sequel we show that the rigidity value of A can be directly derived from the one of C. Let us begin by demonstrating that if C is a rigid type then so is A.

Proof$_1$: Because the generalization set (GS) is complete, we have that ext(/A,w) = ext(B,w) $\cup$ ext(C,w) and that ext(B,w) $\cap$ ext(C,w) = $\varnothing$ for all w $\in$ W. Thus, suppose an instance x of /A. We have that x $\in$ ext(/A,w) and either x $\in$ ext(B,w) or x $\in$ ext(C,w). Suppose that x $\in$ ext(B,w) then the only manner that x can cease to be an instance of /A, i.e., x $\notin$ ext(/A,w') (for a w'$\neq$w) is if x ceases to be an instance of B (x $\notin$ ext(B,w'). Now, since B is rigid then it is necessarily the case that x $\in$ ext(B,w') (*mutatis mutandis* the same can be shown if x $\in$ ext(C,w)). □

Now, suppose that C is not rigid, i.e., that C is non-rigid (either anti-rigid or semi-rigid). In this case, we can show that A is necessarily semi-rigid.

Proof$_2$: Because the GS is complete, we have that ext(/A,w) = ext(B,w) $\cup$ ext(C,w) and that ext(B,w) $\cap$ ext(C,w) = $\varnothing$ for all w $\in$ W. Since C is non-rigid then there is an x such that x $\in$ ext(C,w) and x $\notin$ ext(C,w') for some arbitrary w,w' $\in$ W. Because C is a subtype of /A then we have that x $\in$ ext(/A,w). Now, suppose that /A is rigid. In this case, we have also that x $\in$ ext(/A,w'). Since GS is complete then either x $\in$ ext(B,w') or x $\in$ ext(C,w'). Since the latter is not the case, we must have that x $\in$ ext(B,w'). However, since B is rigid then we also have that x $\in$ ext(B,w) which is a contradiction since ext(B,w) $\cap$ ext(C,w) = $\varnothing$ for all w. We must then conclude that /A is necessarily non-rigid, which means that /A is either anti-rigid or semi-rigid; /A cannot be anti-rigid since B is rigid and an anti-rigid type cannot be a supertype of a rigid type, ergo, we have that /A is semi-rigid. □

The two patterns arising from the proofs 1 and 2 above are depicted in figures 3.a and 3.b below, respectively.

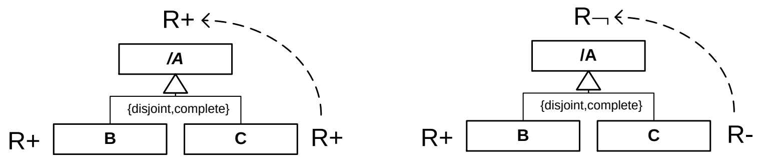

Fig.3. Patterns for inferring the rigidity value of type /A following proofs 1 (a-left) and 2 (b)

Let us suppose now that B is anti-rigid. In this case, if C is rigid then we fall in the pattern of figure 3.b with classes B and C inverted (anti-rigidity is a special case of non-rigidity). If C instead is anti-rigid then A can be either rigid or anti-rigid. Suppose that /A is rigid then it means that its extension is invariant. Since B is anti-rigid then all instances it "acquires" or "looses" must come or go to the complement of B w.r.t. /A, namely, C. Finally, we know that /A cannot be semi-rigid. *By absurdum*, suppose the contrary, i.e., that /A is semi-rigid. In this case, by definition, /A must have a subtype which is rigid. Since neither B nor C are rigid then there must exist a rigid type D (which is a subtype of either B or C). However, a rigid type cannot be a subtype of anti-rigid one. Therefore, we conclude that /A cannot be semi-rigid (***proof 3***). □

Finally, suppose that B is anti-rigid and C is semi-rigid. In this case, /A can be either rigid or itself semi-rigid. We can show, however, that /A cannot be anti-rigid: suppose that /A is anti-rigid. Since C is semi-rigid then there must be at least one rigid subtype D of C. Now, in this case, D would also be a subtype of /A. But since a rigid type cannot be a subtype of an anti-rigid one, we must conclude that /A cannot be semi-rigid (***proof 4***). □

The two patterns arising from the proofs 3 and 4 above are depicted in figures 4.a and 4.b below, respectively.

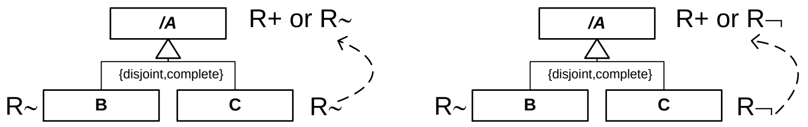

Fig.4. Patterns for inferring the rigidity value of type /A following proofs 3 (a-left) and 4 (b)

Let us suppose now that B is semi-rigid. In this case, if C is rigid then we fall in the pattern of figure 3.b with classes B and C inverted (semi-rigidity is a special case of non-rigidity). If C is instead anti-rigid then we fall in the pattern of figure 4.b with classes B and C inverted. Finally, if C is semi-rigid then the only possibility we can rule out is /A being anti-rigid for the reason already discussed when arguing for pattern of figure 4.b (***proof 5***). □

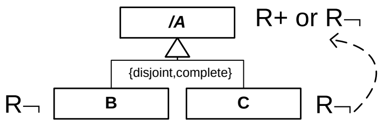

Fig.5. Pattern for inferring the rigidity value of type /A following proof 5

Table 1 below summarizes the results of this section. Table 2 presents a more compact version with the same information.

Table 1. Inferred rigidity value of derived type /A from the different rigidity values of types B and C (painted rows mark cases in which the value of /A is completely determined)

B	C	/A (derived)
R+	R+	R+
R+	R~	R¬
R+	R¬	R¬
R~	R+	R¬
R~	R~	R+, R~
R~	R¬	R+, R¬
R¬	R+	R¬
R¬	R~	R+, R¬
R¬	R¬	R+, R¬

Table 2. Summarized version of Table 1

B	C	/A (derived)
R+ (i.e., for whatever rigid type)	R+	R+
R+	R- (i.e., for whatever non-rigid type)	R¬
R¬ (i.e., for whatever semi-rigid type)	R-	R+, R¬
R~ (i.e., for whatever anti-rigid type)	R~	R+, R~

3.2. Derivation by Exclusion

We now analyze the case of *derivation by exclusion*, i.e., we analyze the case of the derived type /C (following the pattern of figure 2.a) such that the instances of /C are exactly the instances of A which are not instances of B. Once more, here we can show that the rigidity value of /C can be derived from the ones of B and A.

We start by fixing the rigidity value of B as a rigid type. Firstly, let us show that if A is rigid then so is /C (figure 6.a).

Proof$_6$: If A and B are rigid then theirs extension is invariant, i.e., for all w,w' ∈ W, ext(A,w) = ext(A,w') and ext(B,w) = ext(B,w'). Now, since ext(A,w) = ext(B,w) ∪ ext(/C,w), for all w ∈ W, we must conclude that the extension of C is also world invariant, i.e., ext(/C,w) = ext(/C,w'), for all w,w'. Alternatively, we can assume that /C is non-rigid. If this is the case there are x,w,w' so that x ∈ ext(/C,w) and x ∉ ext (/C,w'). But since /C is a subtype of A then we know that x ∈ ext(A,w) and x ∈ ext(A,w'). Since this generalization set is complete, we have that x ∈ ext(B,w'). Now, since B is rigid then it is also the case that x ∈ ext(B,w). But having that x is an instance of both B and /C in w, and that B and /C are disjoint, we have a contradiction, thus, concluding that /C cannot be non-rigid, i.e., it must be rigid. □

Let us now take the case that A is semi-rigid (notice that A cannot be anti-rigid). In this case, /C must be non-rigid (figure 6.b).

Proof$_7$: Suppose that /C is rigid then we have a pattern which is analogous to the one in figure 3.a. Notice that the only assumption we make in proof 1 is that the generalization set involving A, B and C is a disjoint and complete one. So, here again we have if B and C are rigid so is A which is a contradiction (our initial assumption is that A is semi-rigid). Thus, /C must be non-rigid. □

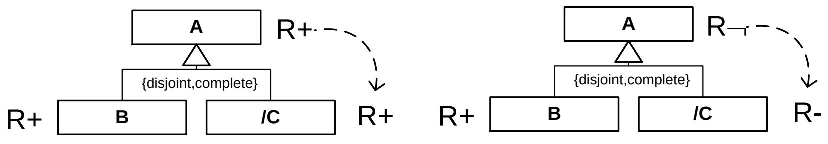

Fig.6. Patterns for inferring the rigidity value of type /C following proofs 6 (a-left) and 7 (b)

Let us now suppose that B is anti-rigid. Here, once more, the rigidity value of /C can be derived from the one of A: if A is rigid then /C must be non-rigid; if A is semi-rigid then /C must be either rigid or semi-rigid; if A is also anti-rigid then so is /C. In the latter case, however, the model must be deemed incomplete.

Proof$_8$: Since B is anti-rigid then there is an x as well as w,w' ∈ W so that x ∈ ext(B,w) and x ∉ ext(B,w'). Since B and /C are disjoint we know that x ∉ ext(/C,w), and since B is a subtype of A we know that x ∈ ext(A,w). Now, since A is rigid then its extension is invariant and then we have that x ∈ ext(A,w'). Since this generalization set is complete, we have that ext(A,w') = ext(B,w') ∪ ext(/C,w'). Given that x ∉

ext(B,w') we must have that ext(/C,w'). This demonstrates that /C's extension is not invariant and, thus, that it cannot be a rigid type. Ergo, /C is non-rigid. □

Proof$_9$: If A is semi-rigid then (by definition) A must have as a subtype a rigid type. Since the Generalization set involving B and /C (and having A as common supertype) is complete then either B or one of its subtypes is rigid or /C or one of its subtypes is rigid. B is anti-rigid and its subtypes cannot be rigid (an anti-rigid type cannot be a supertype of a rigid one). Thus, either /C is rigid or one its subtypes is rigid. If /C is not rigid then the one manner one of its subtypes can be rigid is if /C is semi-rigid. We conclude then that /C is either Rigid or semi-rigid. □

Proof$_{10}$: If A is anti-rigid then /C cannot be rigid since an anti-rigid type cannot be a supertype of a rigid one. Moreover, /C cannot be a semi-rigid type because (by definition) /C would have as subtype a rigid type D. Now, in this case, D would be a (indirect) subtype of A and we would have an anti-rigid type as a supertype of a rigid one. Thus, we must conclude that C must be anti-rigid in this case. □

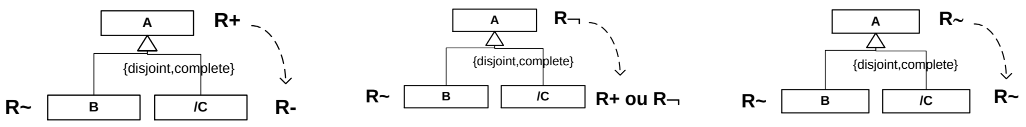

Fig.7. Patterns for inferring the rigidity value of type /C following proofs 8 (a-left), 9 (b-center) and 10 (c-right)

Finally, let us now suppose that B is semi-rigid. Notice that, in this case, A cannot be anti-rigid. This is because by definition B must have a rigid subtype which would in turn be a (indirect) subtype of A. However, an anti-rigid type cannot be a supertype of a rigid one. If A is itself semi-rigid then there is nothing else at this point that we can infer regarding the rigidity value of /C, since it can be rigid, anti-rigid or semi-rigid (***proof 11***). □

Moreover, we can also prove now is that if A is rigid then /C must be non-rigid.

Proof$_{12}$: Since B is semi-rigid then there is an x as well as w,w' ∈ W so that x ∈ ext(B,w) and x ∉ ext(B,w'). Since B and /C are disjoint we know that x ∉ ext(/C,w), and since B is a subtype of A we know that x ∈ ext(A,w). Now, since A is rigid then its extension is invariant and then we have that x ∈ ext(A,w'). Since this generalization set is complete, we have that ext(A,w') = ext(B,w') ∪ ext(C,w'). Given that x ∉ ext(B,w') we must have that ext(/C,w'). This demonstrates that /C's extension is not invariant and, thus, that it cannot be a rigid type. Ergo, C is non-rigid. □

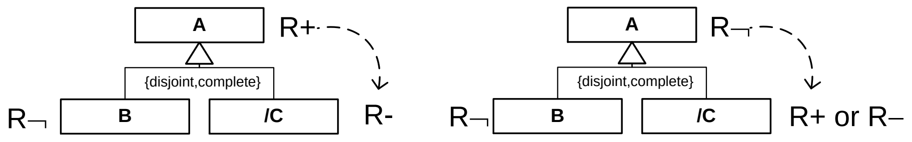

Fig.8. Patterns for inferring the rigidity value of type /C following proofs 11 (a-left) and 12 (b)

The results of this section are summarized in table 3 below.

Table 3. Inferred rigidity value of the derived type /C from the different rigidity values of types B and A (painted rows mark cases in which the value of /C is completely determined)

	B	**A**	**/C (derived)**
1	R+	R+	R+
~~2~~	~~R+~~	~~R~~~	~~Logically Inconsistent~~
3	R+	R¬	R-
4	R~	R+	R-
5	R~	R~	R~
6	R~	R¬	R+, R¬
7	R¬	R+	R-
~~8~~	~~R¬~~	~~R~~~	~~Logically Inconsistent~~
9	R¬	R¬	R+, R-

4 Deriving Ontological Categories of Object Types in OntoUML

In this section, by crossing the results of section 3 with the ontological categories underlying OntoUML, we demonstrate how the ontological category of Object Derived types in OntoUML can be inferred from the ontological categories of the types participating in the derivation.

4.1. Derivation by Union

In table 4 below, we analyze the possibilities of ontological categories that can take the place of classes B, C and /A in the first row of table 2. In the sequel, we analyze the second row of table 2. For the sake of limitation, we will limit our analysis here to only these two cases.

Table 4. Inferred ontological category of derived type /A from those of types B and C

	B	**C**	**/A (derived)**
1	**Kind**	Kind	Category
2		Subkind	Category
3		Category	Category
4	**subkind**	Kind	Category
5		Category	Category
6		Subkind	~~kind,~~ subkind, category
7	**category**	Kind	Category
8		Subkind	Category
9		Category	Category

As one can see, in eight out of nine valid possibilities for the configuration of types B and C we can directly infer the ontological category of /A. Since an object cannot be instance of more than one Kind, a Kind cannot have a supertype another Sortal. Thus, whatever type which is a supertype of a Kind must be a Characterizing type. Since an anti-rigid type cannot be an instance of a rigid type, we have that a supertype of a Kind must either be a Category or a Mixin. Since in row 1 of table 2 the derived type is rigid, we conclude that the derived type /A must be a Category (rows 1-3 of table 4). For an analogous reason, in row 4, /A must be a Category (in table 4, row 4 is equivalent to row 2 with columns B and C inverted).

From the constraints that a characterizing type cannot be a subtype of a Sortal type, and that an anti-rigid type cannot be a supertype of rigid one, we have that a Category can only have as a supertype either a Category or a Mixin. From that, we conclude that in rows 5 but also 7-9 of table 4, the type /A must be a Category.

The only case that leaves open three possibilities for /A is the one in row (6) of this table. However, as previously discussed, a kind is a stereotypical case of a root *base type* since it is the type the supplies the principle of identities for its instances. In fact, the set of object instances of a model is exactly the union of the extension of all kinds [1]. As a consequence, the configuration in which /A is a kind *derived by union*, albeit logically consistent, is ontologically inconsistent. Thus, given that B and C are subkinds, the *derived by union* type /A can either be another subkind or a category. Notice that such a model is necessarily incomplete, i.e., the subkinds B and C must have as supertypes kinds B^K and C^K from which they should inherit their principle of identity. Now, the ontological category of /A can be determined by truth value of formula ($B^K = C^K$). In other words, if ($B^K = C^K$) it means that /A is a subkind which is also a subtype of B^K(or C^K); In contrast, if ($B^K \neq C^K$) then by definition /A is the union of entities that obey different principles of identity and is a category.

Let us now analyze the case of row (2) in table 2. Since the only ontological category which is semi-rigid is a *mixin* then we conclude that whatever combination we have of rigid type (kind, subkind or category) playing the role of B in figure 3.b with a non-rigid type (phase, role, roleMixin, phaseMixin, mixin) playing the role of C then the derived type /A (derived by union) must necessarily be a *mixin*.

These conclusions are summarized in table 5 below.

Table 5. A determinate version of Table 5 with additional constraints as model assumptions

	B	C	/A (derived)	Constraints
1	kind	R+	Category	
2	category	R+	Category	
3	subkind	subkind	Subkind	subtype(B,B^K), kind(B^K), subtype(C, C^K), kind(C^K) and ($B^K = C^K$)
4	subkind	subkind	Category	subtype(B, B^K), kind(B^K), subtype(C, C^K), kind(C^K) and ($B^K \neq C^K$)
5	R+	R-	Mixin	

As we have previously discussed, the results of table 5 can be directly implemented in an automated tool to assist the user in configuring generalization sets of derived types in ontologically correct design patterns. For instance, in figure 9, suppose the modeler defines a common derived type (by union) MaleAnimal of subkinds MalePerson and MaleDog, then the tool could automatically infer that MaleAnimal is a Category (following row 4 of table 5). Moreover, if the modeler then defines the common derived type (by union) Animal, derived from MaleAnimal and FemaleAnimal, the tool could then automatically derive that Animal is a Category (not shown in the figure). Finally, if like in our previously mentioned example, we define InsuredItem as derived by union from the kind Car and the subtype of Person named InsuredPerson (a role), then the tool can derive that InsuredItem is a mixin (following row 5 of table 5).

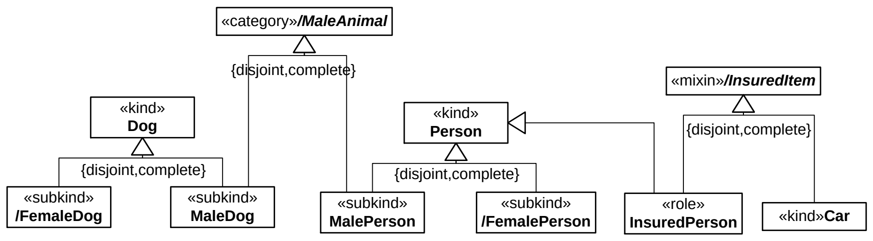

Fig.9. Inferring the Ontological Categories of Derived Object types in OntoUML for the cases of Derivation by Union and Derivation by Exclusion

4.2. Derivation by Exclusion

Here, once more, due to the lack of space, we will focus only on the first row of table 3. In table 6 below, we analyze the possibilities of ontological categories that can take the place of classes A, B and /C in that row.

Table 6. Inferred ontological category of derived type /C from those of types A and B

	A	**B**	**/C (derived)**
1	**kind**	Subkind	Subkind
2	**subkind**	Subkind	Subkind
3	**category**	Kind	kind, subkind, category
4		Subkind	kind, subkind, category
5		Category	kind, subkind, category

As one can see, in the case that A is a kind, the only real possibily for B is to be a subkind since: a kind cannot be a supertype of another kind, and a kind cannot be a supertype a category. In that case, and for the same reasons, /C must also be a subkind. In a similar manner, if A is a subkind then B must also be a subkind, since: a subkind cannot be a supertype of a kind, and a subkind cannot be a supertype a category (a sortal cannot be a supertype of a characterizing type). For the same reasons, /C must be a subkind. An example of situation exemplifying row 1 of table 6 can be found 1 figure 9. If the modeler derives /FemalePerson by exclusion of subkind MalePerson w.r.t. to the kind Person then the tool can automatically infer that /FemalePerson must be a subkind.

In case A is a category, the situation is much less definitive. Since both B and /C could, in principle, assume any possibility among rigid types. In that case, we again would need to use information about other types in the model to draw a more conclusive inference. For instance, if we the modeler declares the model to be complete (an operation common to a particular mode of validation in OntoUML [11]), then we could infer that both B and /C would be kinds. This is because: every subkind in the model must be a subtype of a unique kind and every characterizing type in the model must be specialized (directly or indirectly) by a kind. In other words, if the model is assumed to be complete then neither B or /C could be either subkinds or categories.

5 Final Considerations and Future Work

In this paper, we have presented a theoretical contribution that can be of value both to traditional conceptual modeling and domain ontology engineering. On one side, it allows for employing ontological meta-properties which are well-known in the ontology engineering literature (namely, rigidity and its associated notions) to be used to define finer-grained distinctions among the categories of object types in conceptual modeling. On the other hand, it allows for employing patterns for object type derivation which are well-known in conceptual modeling to be used in the definition of taxonomic structures in ontology engineering.

In particular, the first contribution of this paper is to formally prove a number of patterns that demonstrate how the rigidity value of derived object types can be inferred from the rigidity value of other types participating in the associated derivation rules. As demonstrated in a number of works [7-9], the identification of the rigidity value of object types play an important role in evaluating the quality of taxonomic structures. These patterns, as presented in section 3, can be used in any approach in which these ontological meta-properties are recognized. Noticeable examples are OntoClean [8,9], ORM 2.0 [10] and OntoUML [1,7].

As a second contribution of this paper, we show the practical relevance of these inference patterns by demonstrating how they can be applied in the automated support for conceptual model construction in OntoUML. In section 4, we demonstrate how the ontological categories of derived object types in OntoUML can be inferred from the ontological categories of the types participating in the derivation.

Due to space limitations, we had to make a number of successive selections here. Firstly, we have only addressed two of Olive´s derivation patterns. Secondly, a in adapting the inference patterns of section 3 to OntoUML, we have merely addressed two cases for *Derivation by Union* and one case of *Derivation by Exclusion*. Finally, again due to space limitations, we refrained from showing that although exemplified by generalization sets with only two subtypes, the proofs presented here can be generalized to generalizations sets with any number of types. In an extended version of this paper, we intend to complete all the aforementioned gaps. Moreover, with regards to the adaptation of the full set of inference patterns to OntoUML, we intend to present there their implementation in the OntoUML editor [11].

Acknowledgments. This research is supported by FAPES (PRONEX grant #52272362) and CNPq (productivity grant #311578/2011-0).

References

1. Guizzardi, G. Ontological Foundations for Structural Conceptual Models, Universal Press, The Netherlands, 2005. ISBN 1381-3617.
2. Guizzardi, G.; Lopes, M.; Baião, F.; Falbo, R. On the importance of truly ontological representation languages, International Journal of Information Systems Modeling and Design (IJISMD), 2010. ISSN: 1947-8186.

3. Carolo, F., Burlamaqui, L., Improving Web Content Management with Semantic Technologies, Semantic Technology Conference (SemTech), San Francisco, 2011.
4. Object Management Group, Semantic Information Model Federation (SIMF): Candidates and Gaps, online: http://www.omgwiki.org/architecture-ecosystem/.
5. Bauman, B. T., Prying Apart Semantics and Implementation: Generating XML Schemata directly from ontologically sound conceptual models, Balisage Markup Conference, 2009.
6. U.S. Department of Defense, Data Modeling Guide (DMG) for an Enterprise Logical Data Model (ELDM), available online: http://www.omgwiki.org/architecture-ecosystem/lib/exe/fetch.php?media=dmg_for_enterprise_ldm_v2_3.pdf.
7. Guizzardi, G., Wagner, G., Guarino, G., van Sinderen, M., An Ontologically Well-Founded Profile for UML Conceptual Models, 16th International Conference on Advanced Information Systems Engineering (CAISE 2004), Riga.
8. Guarino, N.; Welty, C., Evaluating Ontological Decisions with OntoClean, Communications of the ACM, 45(2), pp.61-65, 2002.
9. Guarino, N.; Welty, C., An Overview of OntoClean, in S. Staab, R. Studer (eds.), Handbook on Ontologies, Springer Verlag, 2010.
10. Halpin, T., Morgan, T., Information Modeling and Relational Dababases, Morgan Kaufman, 2008. ISBN 1558606726
11. Benevides, A.B.; Guizzardi, G. A Model-Based Tool for Conceptual Modeling and Domain Ontology Engineering in OntoUML, LNBPI, Vol. 24, 2009. ISSN 1865-1356.
12. Benevides, A.B. et al., Validating modal aspects of OntoUML conceptual models using automatically generated visual world structures, Journal of Universal Computer Science, Vol. 16/20, Special Issue on Evolving Theories of Conceptual Modeling, 2010.
13. Guizzardi, G., das Graças, A., Guizzardi, R.S.S., Design Patterns and Inductive Modeling Rules to Support the Construction of Ontologically Well-Founded Conceptual Models in OntoUML, 3rd Intl. Workshop on Ontology Driven Inf. Systems (ODISE 2011), London.
14. Olive, A., Conceptual Modeling of Information Systems, Springer-Verlag, 2007.
15. Wieringa, R.J. de Jonge, W., Spruit, P.A.: Using dynamic classes and role classes to model object migration. Theory and Practice of Object Systems, 1(1), 61-83, 1995.