Ontology-Based Model Abstraction

Giancarlo Guizzardi*, Guylerme Figueiredo†, Maria M. Hedblom* and Geert Poels‡
*Conceptual and Cognitive Modeling Research Group (CORE), Free University of Bozen-Bolzano, Bolzano, Italy
†Ontology and Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo, Vitoria, Brazil
‡Faculty of Applied Economics, Ghent University, Belgium
gguizzardi@unibz.it, gvsfigueiredo@inf.ufes.br, mhedblom@unibz.it, geert.poels@ugent.be

***Abstract*—In recent years, there has been a growth in the use of reference conceptual models to capture information about complex and critical domains. However, as the complexity of domain increases, so does the size and complexity of the models that represent them. Over the years, different techniques for complexity management in large conceptual models have been developed. In particular, several authors have proposed different techniques for *model abstraction*. In this paper, we leverage on the ontologically well-founded semantics of the modeling language OntoUML to propose a novel approach for model abstraction in conceptual models. We provide a precise definition for a set of Graph-Rewriting rules that can automatically produce much-reduced versions of OntoUML models that concentrate the models' information content around the ontologically essential types in that domain, i.e., the so-called *Kinds*. The approach has been implemented using a model-based editor and tested over a repository of OntoUML models.**

***Index Terms*—Model Abstraction, Complexity Management in Conceptual Modeling, Ontology-Based Conceptual Modeling**

I. Introduction

In recent years, there has been a growth in the use of reference conceptual models to capture information about complex and critical domains. These models play a fundamental role in different types of critical semantic interoperability tasks. Therefore, it is essential that domain experts are able to understand the subtleties of their content and reason with them. In other words, it is important that conceptual models remain cognitively tractable. However, as complexity of the information about the represented domain grows, so does the size and complexity of the artifacts that represent them.

Over the years, different techniques for complexity management in large conceptual models have been developed. In particular, a number of authors have proposed different techniques for *model abstraction*. These techniques aim at systematically producing reduced versions of the original conceptual models omitting details while concentrating on the gist of the semantic content at hand. However, since basically all traditional techniques have been produced for ontologically neutral conceptual modeling languages (e.g., ER, UML), they are constrained to rely solely on syntactical properties of the models and, more specifically, on topological properties of the graph. This happens because these languages treat all types as being the same, semantically overloading one single construct (e.g., Class or Entity Type) by using it to represent different types of ontological categories.

In contrast, ontology-driven conceptual modeling (ODCM) languages are systematically designed to conform to an underlying ontological theory. In particular, an ODCM language contains exactly the modeling primitives that are necessary to represent the ontological distinctions put forth by its underlying ontology. For example, as different ontological categories of types (e.g., Kinds, Mixins, Roles) play different roles with respect to their instances regarding issues such as classification (e.g., dynamic versus static) and identity, these distinctions should be explicitly represented by the language's constructs.

ODCM approaches have enjoyed an increasing adoption by the Conceptual Modeling community as a number of independent results consistently show their benefits for improving the quality of conceptual models [3], [21], [28]. An example of an ODCM language is OntoUML [12]. It has been successfully employed in academic, industrial and governmental settings to create conceptual models in a number of different domains, including Geology, Biodiversity Management, Organ Donation, Petroleum Reservoir Modeling, Disaster Management, Context Modeling, Datawarehousing, Enterprise Architecture, Data Provenance, Measurement, Logistics, Complex Media Management, Telecommunications, Heart Electrophysiology, among many others [14]. In fact, research shows that it is among the most used ODCM languages in the literature [27]. Moreover, empirical evidence shows that OntoUML significantly contributes to improving the quality of conceptual models without requiring an additional effort to produce them. For instance, the work of [28] reports on a modeling experiment conducted with 100 participants in two countries showing the advantages (in these respects) of OntoUML when compared to a classical conceptual modeling language (EER).

In this paper, we leverage on the ontologically well-founded semantics of OntoUML to propose an approach for model abstraction in conceptual models. We provide a precise definition for a set of Graph-Rewriting rules that can automatically produce much-reduced versions of OntoUML models. These reduced models concentrate the models' information content around the ontologically essential types in that domain, i.e., the so-called *Kinds*. The approach has been implemented using a model-based editor and tested over a varied set of OntoUML models.

The remainder of the paper is organized as follow. Section II discusses related work in the area of complexity management of conceptual models; Section III briefly presents the OntoUML language and its ontological foundations; Section IV presents the main contribution of this paper by defining a set of Graph-Rewriting abstraction rules; Section V presents a plug-in to a model-based editor that implements the proposed abstraction rules. The implementation of this plug-in itself counts as a first validation of the approach (claim to *practical realizability*). However, in that same section, we also report on a study conducted with a varied set of OntoUML models to assess both the proposed rule set and its implementation; Section VI systematically compares the results presented here with relevant related work; Finally, Section VII concludes the paper with some final considerations.

II. Complexity Management of Conceptual Models

The discipline of *complexity management of large conceptual models* (henceforth CM-CM) has been around for quite some time and has been represented in the literature by a series of different approaches and techniques. In fact, [29] claims that "one of the most challenging and long-standing goals in conceptual modeling... is to understand, comprehend and work with very large conceptual schemas".

The challenge and importance of this discipline lies in the following. On one hand, real information systems often have large and extremely complex conceptual models [29]. On the other hand, this complexity poses a serious additional challenge in the comprehension and, consequent, quality assurance of these models. For example, [22] reports on an empirical study conducted with a large and professionally constructed conceptual model.[1] In that study, the authors managed to show that the model contained 879 occurrences of error-prone structures (anti-patterns), 52.56% of which really introduced representation errors according to the creators of the model. A more dramatic case (both in size and percentage of occurrences of anti-patterns) is reported in [4], in which the authors show a case of an anti-pattern present in 85% of the relevant relations in Wikidata.

According to [29], the methods for CM-CM can be classified in three areas, namely, *Clustering Methods*, *Relevance Methods*, and *Summarization Methods*. Clustering is about classifying the elements of a conceptual model into groups, or clusters, according to some criteria (e.g., a similarity function); Relevance Methods are about the application of ranking functions to the elements of a model in order to obtain ordered lists (i.e., a ranking) of model elements according to their perceived relevance in representing the domain at hand; finally, Model Summarization is about producing from an original model a reduced version consisting only of the elements that are judged to be of more relevance for representing the domain at hand. A variation of Model Summarization is Model Abstraction, which can be described as "a process that transforms lower-level elements into higher-level elements containing fewer details on a larger granularity" [7]. The idea here is to allow for the possibility of "zooming out" of models in order to provide the user with the "bigger picture" of the domain at hand, allowing her to focus on the gist of the model by filtering out specific details [7]. In clustering methods, the goal is to break down a model in fragments such that the sum of these fragments should be informationally equivalent to the whole (i.e., to the original model). In contrast, relevance and summarization methods (including model abstraction) aim to produce partial views of the original model at hand. In other words, while clustering methods have *lossless model transformations*, the latter classes of methods are based on *lossy transformations*.

Given the purpose of this paper, we henceforth focus our discussion in this class of lossy methods and, in particular, in summarization/abstraction methods. First, however, we elaborate on a drawback that is common to the majority of existing methods in all these classes. Namely, as nearly all these methods are based on classic conceptual modeling notations (e.g., UML, ER) [29], they are constrained to rely almost exclusively on syntactic (mainly topological) properties of the addressed models. These properties include *closeness* (a quantitative evaluation of the links among elements in the model) [10], *hierarchical distance* (length of the shortest relationship path between entities), *structural-connective distance* (elements are considered closer if they are neighbors in a hierarchy mereological or subtyping structure), or *category distance* (elements are considered to be closer if one subtypes the other) [1]. For example, [5] proposes a (relevance) method based on the assumption that the number of attributes and relations characterizing an element in a model can be used as a (heuristic) measure of its relevance for that model. In the same spirit, [25], [26] go as far as proposing PageRank-style algorithms to infer the relevance of elements in entity-relationship diagrams and RDF schemas (even ignoring the difference between association and subtyping relations). The problem with relying solely on these properties is that there is no guarantee that a model element satisfying some topological requirement (e.g., a node with more edges connected to it) by necessity represents the model's most important concepts. This is related to the work by [18], [19], that while criticizing existing CM-CM methods, referred to it as *lack of cognitive justification*.

One of the main approaches for Model Abstraction in the literature is the one proposed by Egyed in [6]–[8]. It is a tool-supported approach that provides conceptual model designers with the ability to 'zoom out' from a model to investigate and reason about its bigger picture. Like in our case, the proposed approach is based on a set of pattern detection and replacement rules. The approach is built around a classic conceptual modeling technique (UML) and, hence, it heavily relies on structural (topological) properties of the diagram. As a consequence, it is subject to some extent to all the aforementioned drawbacks. In fairness, the author makes a significant effort in defining structural patterns that reflect the intuitive semantics of UML class diagrams, for instances, regarding transitivity of dependence, method calling, inheritance and propagation from parts to wholes. However, the whole approach has a strong focus of the use of these diagrams from the point of view of Object-Oriented Programming and,

[1] This model consisted of 3800 classes, 61 datatypes, 1918 associations, 3616 subtyping relations, 698 generalization sets and 865 attributes, i.e., navigable association ends [22].

hence, many fundamental conceptual modeling primitives are simply ignored. These include not only modal aspects of class instantiation, existential dependence relations, but also more basic aspects such as generalization sets and association classes (or reified associations).

In contrast with all the aforementioned approaches, our proposal in this paper focuses mainly on the *ontological semantics* [12] of the elements represented in a conceptual model. In particular, it produces an abstraction of an original model by representing a summarized version of the domain information around the backbone of the ontological *kinds* of entities represented therein. As supported by a significant body of evidence in cognitive psychology [17], [31], [32], these *kinds* (also termed *substance sortals*) represent the most salient category of types in human cognition, being responsible for our most basic operations of object individuation and identity.

The approach presented here (and detailed in section IV) is only made possible because it is based on a non-classical CM language, namely, the ODCM language OntoUML (briefly presented in section III). There are two CM-CM methods in the literature that are based on the same language, namely, the approaches of [9] and [15], [16]. However, none of these are methods of model abstraction. The former proposes a method for model clustering by viewpoint extraction. The idea is to divide the model into a number of clusters, each of which represents a well-founded *ontological viewpoint*. As a clustering method, the approach produces a set of clusters whose sums are informationally equivalent to the original model (lossless transformation). The latter approach, although also being about view extraction, can be considered as a summarization method, since it produces partial versions of the original model (lossy transformation). It is, nonetheless, not a method of model abstraction. The idea there is that, given a set of entities from the original model provided by the model user, the method produces a subset of that model containing the elements that are more relevant for the context of the selected entities. One important point is that the notion of relevance there is not a syntactic but an ontological one, related to notions such as conservation of identity, of existential dependence, of parthood, etc. In any case, different from our proposal, the approach achieves summarization by presenting a model that is identical to a proper part of the original model, not by producing a global view of the entire model in a much higher level of abstraction.

III. A Brief Introduction to UFO and OntoUML

OntoUML is a language whose meta-model has been designed to comply with the ontological distinctions and axiomatization of a theoretically well-grounded foundational ontology named UFO (Unified Foundational Ontology) [12], [14]. UFO is an axiomatic formal theory based on contributions from Formal Ontology in Philosophy, Philosophical Logics, Cognitive Psychology, and Linguistics. A recent study shows that UFO is the second-most used foundational ontology in conceptual modeling and the one with the fastest adoption rate [27]. In the remainder of this section, we briefly explain a selected subset of the ontological distinctions put forth by the Unified Foundational Ontology (UFO). We also show how these distinctions are represented by the modeling primitives of OntoUML. In order to do that, we rely on the running example depicted in fig. 1. For an in-depth discussion, philosophical justifications, formal characterization and empirical support for these categories one should refer to [11], [12].

Take a domain in reality restricted to endurants [12] (as opposed to events or occurrents). Central to this domain we will have a number of object *Kinds*, i.e., the genuine fundamental types of objects that exist in this domain. The term "kind" is meant here in a strong technical sense, i.e., by a kind, we mean a type capturing essential properties of the things it classifies. In other words, the objects classified by that kind could not possibly exist without being of that specific kind. In fig. 1, we have represented three kinds, namely, 'Person','Car' and 'Organization'. These are the fundamental kinds of entities that are deemed to exist in the domain.

Kinds tessellate the possible space of objects in that domain, i.e., all objects belong necessarily to exactly one kind. However, we can have other static subdivisions (or subtypes) of a kind. These are naturally termed *Subkinds*. As an example, the kind 'Person' can be specialized in the subkinds 'Man' and 'Woman'; the kind 'Organization' also be specialized in subkinds, e.g., 'Car Agency' is a static subkind of *Organization* (fig. 1).

Object kinds and subkinds represent essential properties of objects (they are also termed rigid or static types [12]). We have, however, types that represent contingent or accidental properties of objects (termed anti-rigid types [12]). These include *Phases* and *Roles*. The difference between the contingent properties represented by a phase and a role is the following: phases represent properties that are intrinsic to entities; roles, in contrast, represent properties that entities have in a relational context, i.e., contingent relational properties.

Phases but also typically subkinds appear in OntoUML models forming (disjoint and complete, i.e., exhaustive) *partitions* following a *Dividing Principle* [30]. For example, in fig. 1, we have the following *phase partitions*: the one including 'Living Person' and 'Deceased Person' (as phases of 'Person' and according to a 'life status' dividing principle); the one including 'Child', 'Teenager' and 'Adult' (as phases of 'Living Person' and according to a 'developmental phase'); the one including 'Available Car' and 'Under Maintenance Car'(as phases of 'Car' and according to a 'operational status'). Since they are exclusively composed of phases, these are all dynamic partitions [30]. In this model, we also have a (static) *subkind partition* formed by the subkinds 'Man' and 'Woman', dividing 'Person' according to 'gender'.

Kinds, Subkinds, Phases, and Roles are categories of object *Sortals*. In the philosophical literature, a sortal is a type that provides a uniform principle of identity, persistence, and individuation for its instances [12]. To put it simply, a sortal is either a kind (e.g., 'Person') or a specialization of a kind (e.g., 'Husband', 'Teenager', 'Woman'), i.e., it is either a type representing the essence of what things are or a sub-

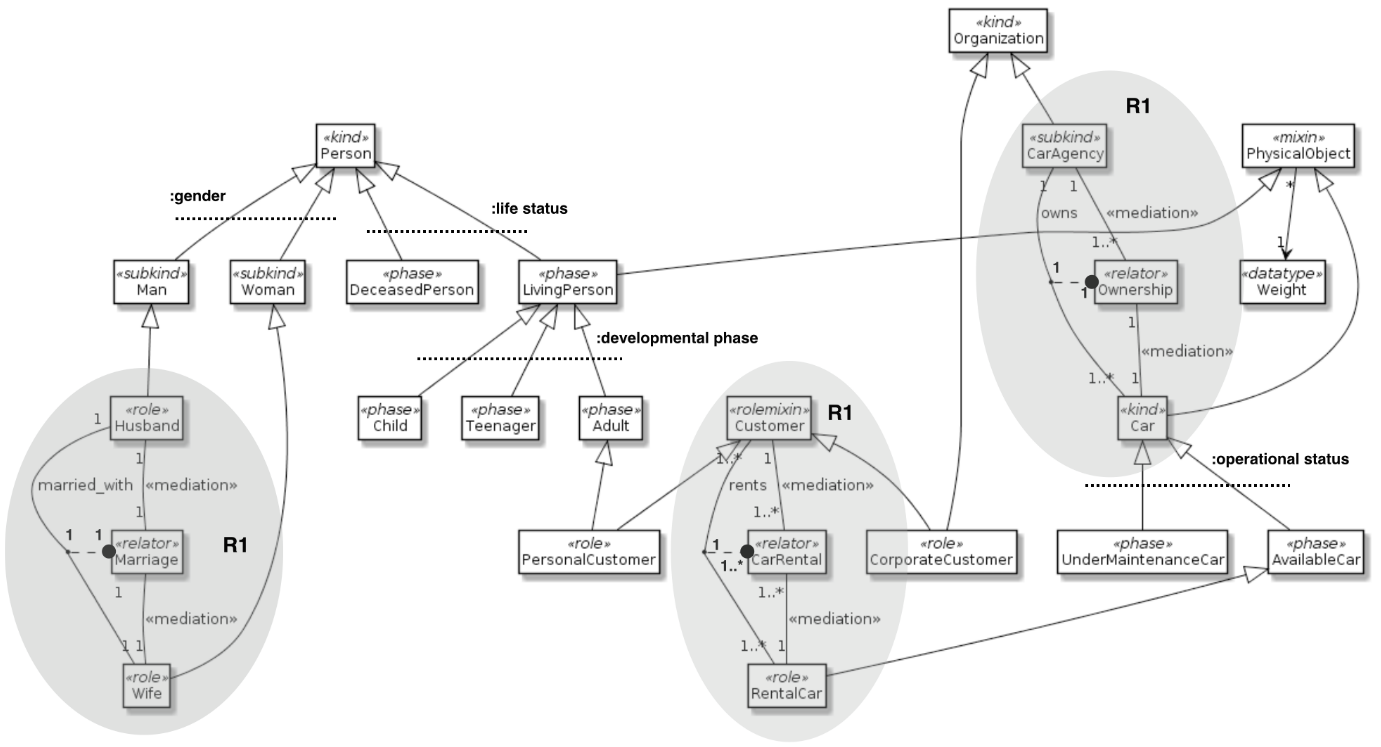


Fig. 1. An Example of an OntoUML Model.

classification applied to the entities that “have that same type of essence”.

Relators (or *relationships* in a particular technical sense [11]) represent clusters of relational properties that “hang together” by a nexus. Moreover, relators (e.g.,‘enrollments’, ‘employments’, ‘presidential mandates’, ‘citizenships’, but also ‘marriages’, ‘Car Rentals’, and ‘Ownerships’ (fig. 1)) are full-fledged *Endurants*. In other words, entities that endure in time bearing their own essential and accidental properties and, hence, first-class entities that can change in a qualitative manner while maintaining their identity. As discussed in depth in [11], [12], relators are fundamental for addressing a number of problems in conceptual modeling, ranging from ambiguity of cardinality constraints, to ambiguity in relation specialization, to transitivity of part-whole relations.

As discussed in depth in [11], relators are the truthmakers of relational propositions, and relations (as classes of n-tuples) can be completely derived from relators [12]. For instance, it is the ‘marriage’ (as a complex relator composed of mutual commitments and claims) between ‘John’ and ‘Mary’ that makes true the proposition that “John is the husband of Mary”. In other words, the pair <John,Mary> is an instance of the relation ‘married-with’ because there is a particular instance of the relation ‘Marriage’ connecting John and Mary (fig. 1). In OntoUML, the relation of *derivation*, connecting relations to the relator types from which they are derived is represented by a dashed line with a black circle in the relator type association end (fig. 1). Technically, Relators bind their relata because they are existentially dependent entities (e.g., the marriage between John and Mary can only exist if John and Mary exist). The formal connection between Relators and their relata is made by the so-called *mediation* relations, a particular type of existential dependence relation [12].

Objects participate in relationships (relators) playing certain *Roles*. For instance, people play the role of spouse in a marriage relationship; a person plays the role of president in a presidential mandate. ‘Spouse’ and ‘President’ (but also typically ‘Student’, ‘Teacher’, ‘Pet’, ‘Rented Car’[2]) are examples of what we technically term a *role* in UFO, i.e., a relational contingent sortal (since these roles can only be played by entities of a unique given kind). There are, however, relational and contingent role-like types that can be played by entities of multiple kinds. An example is the role ‘Customer’ (which can be played by both people and organizations). We call these role-like types that classify entities of multiple kinds *RoleMixins*.

In general, types that represent properties shared by entities of multiple kinds are termed *Non-Sortals*. In UFO, besides rolemixins, we have two other types of non-sortals, namely *Categories* and *Mixins*. Categories represent necessary properties that are shared by entities of multiple kinds (e.g., the category ‘Physical Object’ represent properties of all kinds of entities that have masses, spatial extensions, etc.). In contrast,

[2] In order to illustrate different combinations of cardinality constraints, in fig.1, we assume here that a RentedCar can be related to several rentals provided that they do not overlap in time, hence, the cardinality ‘1..*’ on the side of Car Rental as well as the corresponding cardinality constraints on the derived ‘rents’ relation.

mixins represent shared properties that are necessary to some of its instances but accidental to others. For example, in fig. 1, we have the mixin 'Physical Object', since it represent properties (e.g., having a 'Weight') that are necessary to 'Cars', while being accidental to instances of 'Person' (people are only physical objects when they instantiate the *phase* 'Living Person'). Categories and mixins are, in contrast to rolemixins, considered as Relationally Independent Non-Sortals.

Finally, as all OntoUML models are valid UML models, the classes in the model can also be connected (via particular types of relations termed *navigable association ends*) with datatypes. Datatypes, in UML, are classifiers that include the so-called 'pure values'. As discussed in [12], datatypes are sets of abstract elements. As sets they can be defined by a formula or by extension, in which the total membership of that datatype is explicitly enumerated. In these cases, these datatypes are termed *enumerations*. Still, in all UML models, association ends of relations can be characterized by so-called *rolenames*, which are intended to represent the way an instance of a type (connected to that association end) participates in that relation. Examples of enumerations and rolenames are shown later in this article.

IV. Abstraction Rules

In this section, we present a set of *graph-rewriting rules*.[3] As a system of graph-rewriting, the approach replaces fragments of the original model by reduced fragments that, while filtering out details, maintain essential information. In particular, these rules and their specific order of application (from R1 to R4) have been designed according to the following rationale.

As previously explained, in ODCMs, we have that:

1) objects take a precedence over relators, given that the latter are existentially dependent on the former. So, in the reduced model, we focus on the representation of objects as well as the relations derived from these relators without representing the latter explicitly. In other words, instead of explicitly representing the relators that are truthmakers of certain relational properties involving objects, we simply represent these derived properties. R1 was designed to address this abstraction step and should be the first to be applied to a model;

2) the most important object types in an ODCM are the *Kinds* of entities that exist in that domain, i.e., the rigid identity-supplying types. Once relator types have been abstracted from a model (R1), the goal is to distribute the information in a such a way that it ends up being concentrated around the *kinds* represented in the model. We do that by: firstly, taking all properties of Non-Sortal types and moving it downwards towards the sortals in the model. These non-sortals are then abstracted away in the resulting fragment (R2); secondly, we move all properties of sortals that are not kinds upwards towards the kinds in the model. These (non-kind) sortals are then abstracted away in the resulting fragment (R3); finally, for all subkind and phase partitions, after moving their properties upwards according to (R3), we abstract from these partitions by representing their concrete leaf types in enumeration datatypes (R4). As a result of the application of these rules, the final abstracted model should only contain Kinds and Enumerations, hence, making redundant the use there of the stereotype «kind».

Each of these four rules (R1-R4) occupy a row in Table I. Once more, following the aforementioned rationale, the rules should be applied in this specific order. For each of these rules, whenever a graph structure satisfying the pattern in column *Matching Graph* is found in an OntoUML model, that portion of the model is replaced by the corresponding structure in column *Replacing Graph*. In the remainder of this section, we elaborate on each of these rules by relying on our running example of fig. 1.

A. Abstracting Relators (R1)

As previously discussed, relators are fundamental in ODCMs. For instance, by representing the cardinality constraints $c_1..c_n$, $d_1..d_n$, $a_1..a_n$ and $b_1..b_n$ in table I, we can eliminate instances of the so-called *single-tuple/multiple-tuple cardinality ambiguity problem* [12] in the model; by representing the the cardinality constraints $e_1..e_n$ and $f_1..f_n$, we can eliminate occurrences of the *repeatable instances* anti-pattern as discussed in [22]. However, in order to compress our model, and following the previously discussed rationale, we abstract from this *relator pattern* by replacing each of its occurrences with a structure in which only the derived relations are shown. For example, in the model in fig. 1, there are three occurrences of this pattern (highlighted by oval shadows). By applying R1 to that model, we obtain the model in fig. 2. Hence, for example, instead of having represented there the full relator pattern connecting 'Marriage' to 'Husband' and 'Wife' (as well as the corresponding full set of cardinality constraints and the derivation relation), this pattern is replaced by the graph structure that only represents the 'married-with' and associate roles and cardinality constraints (see the oval marked as R1* in fig. 2).

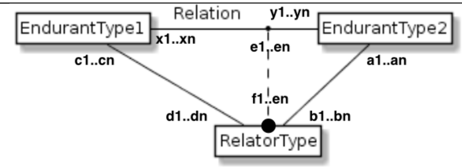
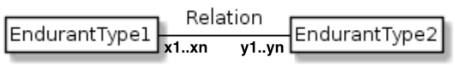
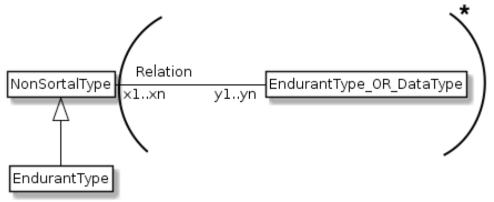
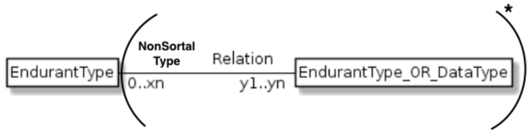
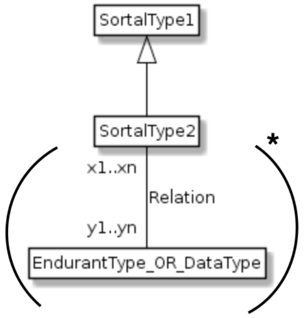
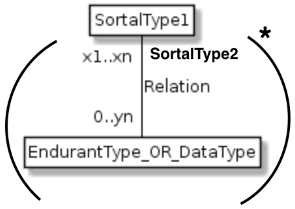
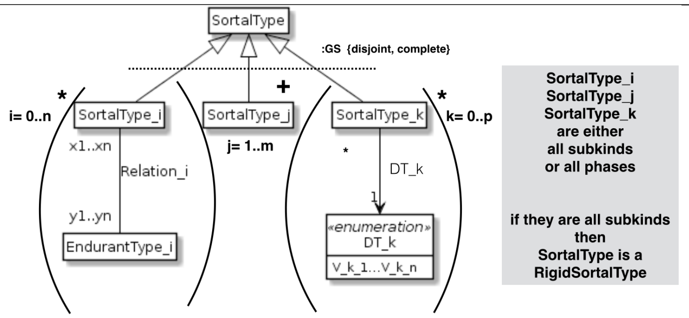
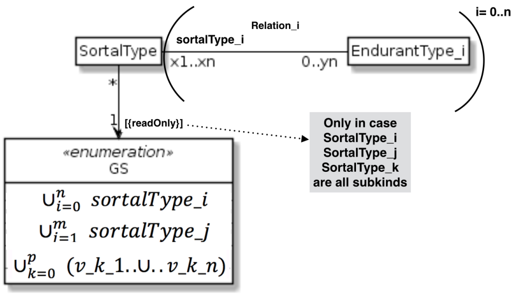
B. Abstracting Non-Sortals (R2)

As discussed in [12], Non-Sortals capture properties that are shared by individuals of different kinds. As discussed there, they play a significant role in *model refactoring*. However, for the sake of model compression, R2 reverses this refactoring operation by replicating the information represented in non-sortal types in its concrete concrete sortal subtypes. This rule serves as an intermediate step for the application of rule R3. In summary, as shown by the rewriting scheme of row 2 in table I, this rule consists of:

1) Abstracting from all Non-Sortal types in the diagram;
2) Replicating all properties of these Non-Sortal in the concrete sortal types that specialize them.

[3] Due to lack of space for this article, we present these rules without employing a specific formal language. In an extension of this paper, we shall present their formalization using the same tools we used for formalizing OntoUML as a Pattern Grammar [33].

TABLE I
GRAPH-REWRITING RULES FOR ONTOUML MODEL ABSTRACTION

Rule	Matching Graph	Replacing Graph
R1	 EndurantType1 Relation y1..yn EndurantType2 x1..xn e1..en c1..cn a1..an f1..en d1..dn b1..bn RelatorType	 EndurantType1 Relation x1..xn y1..yn EndurantType2
R2	 NonSortalType Relation x1..xn y1..yn EndurantType_OR_DataType * EndurantType	 * EndurantType NonSortal Type Relation 0..xn y1..yn EndurantType_OR_DataType
R3	 SortalType1 SortalType2 * x1..xn Relation y1..yn EndurantType_OR_DataType	 SortalType1 * x1..xn SortalType2 Relation 0..yn EndurantType_OR_DataType
R4	 SortalType :GS {disjoint, complete} * + * i= 0..n SortalType_i SortalType_j SortalType_k k= 0..p x1..xn Relation_i j= 1..m * DT_k 1 y1..yn EndurantType_i «enumeration» DT_k V_k_1...V_k_n SortalType_i SortalType_j SortalType_k are either all subkinds or all phases if they are all subkinds then SortalType is a RigidSortalType	 Relation_i i= 0..n sortalType_i SortalType x1..xn 0..yn EndurantType_i * 1 {readOnly} Only in case SortalType_i SortalType_j SortalType_k are all subkinds «enumeration» GS $\cup_{i=0}^{n}$ sortalType_i $\cup_{i=1}^{m}$ sortalType_j $\cup_{k=0}^{p}$ (v_k_1..∪..v_k_n)

More precisely, as demonstrated table I, given a pattern in which we have zero or more occurrences (symbolized there as '*') of a relation R' connecting a non-sortal type NS to another Endurant type T' (or to a datatype D'), we replace this pattern by a structure in which a copy of each R' is created connecting each Endurant Type ST' (direct specializing NS) to T' (or D'). The original non-sortal NS is represented in the replacing structure as a *rolename* connected to the association end attached to each ST'. Although it is true that every T' or D' must associate with at least x_1 instances of NS, for each of the specific subtypes ST', it is not the case that T' or D' must associate with their instances. For this reason, the cardinality constraint x_1 in the original pattern is relaxed to an optional cardinality constraint (i.e., $0..x_n$).

For example, in the model of fig. 2, we have two occurrences of this pattern (highlighted by oval shadows tagged R2). By applying R2 to that model, we obtain the model of fig. 3. So, for example, in fig. 3, the property of having a 'weight' (R' above) connects 'Physical Object' (NS) to the datatype 'Weight'(D' above). This property is then now replicated in the specific concrete types of physical objects, namely, instances of 'Living Person' (ST_1) and 'Car' (ST_2) (see oval marked as R2* in fig. 3). Moreover, the *rolename* 'Physical Object' is attached to the association end connected to each of these sortal types.[4] Finally, the minimum cardinality constraint for each of these association ends are relaxed to zero. After all, given a particular weight, it is neither necessarily associated with a 'Living Person' nor with a 'Car'.

C. Abstracting Sortals (R3)

From an ontological point of view, the most important types of any given domain are the so-called *Kinds* (also called *Substance Sortals*), given that: these are types that provide a principle for identity, individuation and persistence for their

[4] This represents that it is *qua-physical-object* that cars and living people have weights.

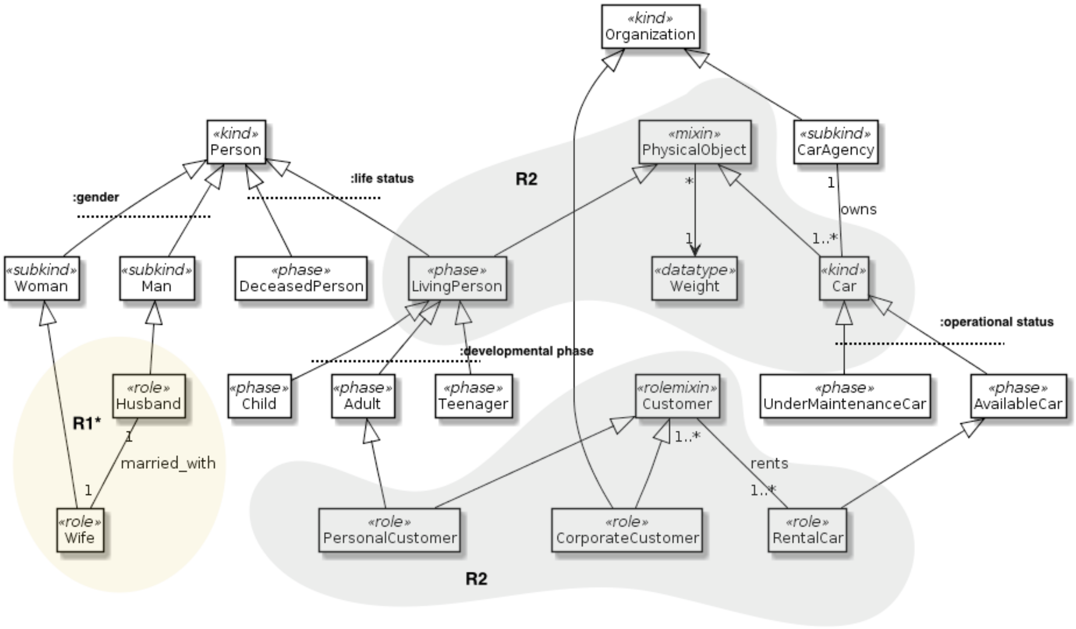


Fig. 2. Model produced as a result of the application of R1 on the model in fig. 1.

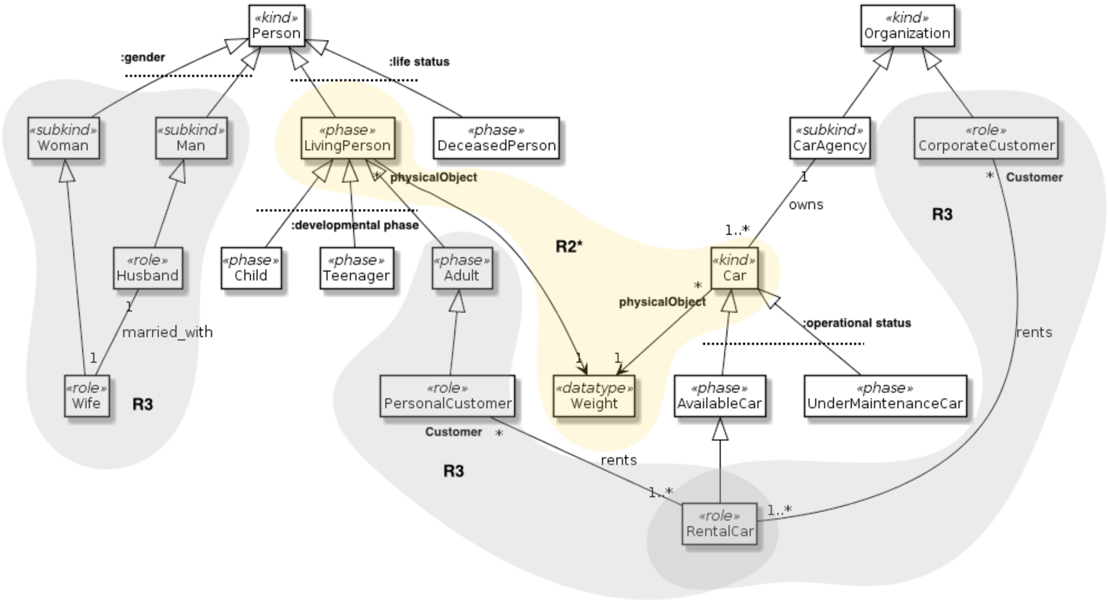


Fig. 3. Model produced as a result of the application of R2 on the model in fig. 2.

instances; they classify their instances necessarily, i.e., in all possible situations (in a modal sense) [12]. As previously discussed, the objective of the R3 is to (through one or more iterative application of this rule) abstract from all other sortals in a model (i.e., subkinds, phases and roles), concentrating the information about the domain around these kinds.

In summary, as shown by the rewriting scheme of row 3 in table I, this rule consists of:

1) Eliminating all non-substance sortal types from the diagram;
2) Moving the properties of these non-substance sortal types to their sortal supertype, until these properties are eventually used to characterize the *kinds* in the model.

More precisely, as demonstrated in table I, given a pattern in which we have zero or more occurrences (symbolized there as '*') of a relation R' connecting a sortal type ST to another Endurant type T' (or to a datatype D'), we replace this pattern by a structure in which a copy of each R' is created connecting each sortal type ST' (direct supertyping ST) to T' (or D'). The original sortal ST is represented in the replacing structure as

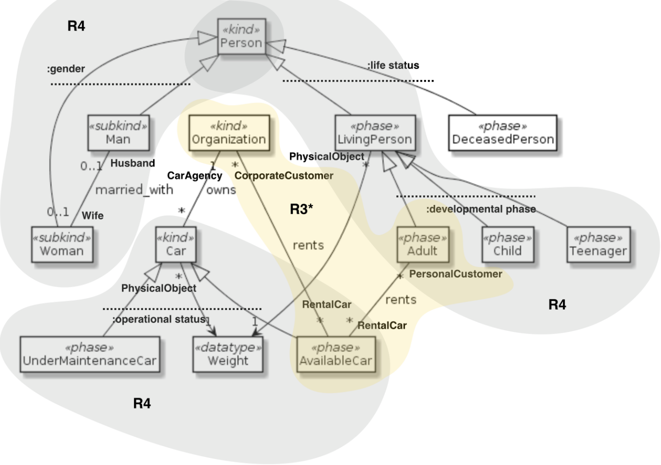


Fig. 4. Model resulting from the application of R3 on the model in fig. 3.

a *rolename* connected to the association end attached to each ST'. Although it is true that instances of ST must be associated with at least y_1 instances of T' (or D'), it is not the case that instances of ST' in general must satisfy the same constraint. For this reason, the cardinality constraint y_1 in the original pattern is relaxed to an optional cardinality constraint (i.e., $0..y_n$).

For example, in the model of fig. 3, we have initially three occurrences of this pattern. By applying R3 to that model, we obtain the model of fig. 4. So, for example, in fig. 3, the relation 'rents' (R' above) connects the sortal 'Personal Customer' - a role played by an 'Adult Living Person' when renting a car (ST_1) - to another sortal 'Rental Car' - a role played by an 'Available Car' when being rented by a customer (ST_2). Due to the application of R3, this relation is then now moved to connect a more abstract supertype of 'Personal Customer' ('Adult' - ST'_1) and a more abstract supertype of 'Rental Car' ('Available Car'- ST'_2) (see oval marked as R3* in fig. 4). Moreover, the *rolenames* 'Personal Customer' and 'Rental Car' are attached to the association ends connected to 'Adult' and 'Available Car', respectively.[5] Finally, the minimum cardinality constraint for each of these association ends are relaxed to zero. After all, it is neither the case that every 'Adult' rents at least one car nor the case that every 'Available Car' must be rented. Of course, as illustrated in R3* in fig. 4, an analogous case can be made for the abstraction of 'Corporate Customer' into the supertyping sortal 'Organization'.

D. Abstracting Subkind and Phase Partitions (R4)

As previously discussed, phases always appear in ODCMs as parts of the so-called *phase partitions*. This is often the case also with multiple subkinds of a given kind, hence, forming *subkind partitions* [13]. Both these partitions define disjoint and complete *generalization sets* over a common supertype. The difference between these partitions is the following. Since phases are anti-rigid types, the phase partitions are dynamic, i.e., the instances of a phase (e.g., 'Child') in a partition (e.g., 'developmental stage') can always possibly (in a modal sense) move to another phase in the same partition (e.g., 'Adult' or 'Teenager'). In contrast, since subkinds are rigid, instances of a subkind (e.g., 'Man') in a given subkind partition (e.g., 'gender') cannot move to other subkinds in the same partition, i.e., subkind partitions are static.

Rule 4 is designed to abstract away non-substance sortals in these partitions by: (i) eventually mo all their propvingerties to their unique supertyping kind (through one or more iterated applications of this rule); (ii) represent all leaf types in these partitions as values in a *enumeration* datatype eponymous to that partition. In summary, as shown by the rewriting scheme of row 4 in table I, this rule consists of:

1) Moving all the properties of a SortalType ST' participating in a partition to the common sortal supertype K in that partition;[6]
2) Transforming each leaf sortal type ST' in a partition into an item in a new enumeration datatype that has the same name as the partition at hand. This enumeration is then related to the common immediate supertype in that partition (K) via a datatype relation D_R (navigable association end in (Onto)UML);
3) If a SortalType being abstract by this rule is already related to another existing enumeration E' then all elements of this enumeration will be included as elements of the new enumeration E created by this rule.

More precisely, as demonstrated table I, we start with a pattern in which we have a generalization set GS containing a number of (all rigid or all anti-rigid) sortal types that share a common (sortal) supertype *SortalType*. Then in GS we have one or more occurrences (symbolized there as '+') of sortal types $SortalType_j$ as well as zero or more occurrences (symbolized there as '*') of sortal types $SortalType_i$ and $SortalType_k$. We then create an enumeration datatype called GS such that all sortal types $SortalType_j$ and $SortalType_i$ appear as elements of the newly created enumeration datatype GS. As for the sortal types $SortalType_k$, which are connected to their own enumeration datatypes, a special treatment is reserved as explained below. We also create a datatype relation D_R connecting *SortalType* to GS. Given that the original generalization set is *disjoint* and *complete*, the cardinality constrains on the association end connected to GS via D_R must be of exactly 1 (minumum cardinality of 1 due to completeness and maxinum cardinality of 1 due to disjointness). Iff the original generalization set GS represents a subkind partition then the association end connected to enumeration GS via D_R must be declared immutable (*readOnly* in (Onto)UML) and, hence, its value must be the same in all possible worlds.[7]

[5] Again, this represents that it is *qua-personal customer* that an adult person rents a car and, analogously, it is *qua-rental car* that an available car participates in a renting relation with a person.

[6] Notice that in, this sense, R4 contains a special variant of R3 for abstracting sortals that are part of a partition.

[7] Following OntoUML's grammatical rules, these sortal type partitions are either exclusively composed of phases or of subkinds.

Moreover, suppose that in the generalization set GS we have one or more occurrences of a sortal types $SortalType_i$ bearing a relation $relation_i$ to another endurant type $EndurantType_i$. In this case, we replace this pattern by a structure in which a copy of each $relation_i$ is created connecting the common (direct) supertype *SortalType* in that partition to $EndurantType_i$. If there are *rolenames* connected to the original sortal $SortalType_i$, these are represented in the replacing structure as *rolenames* connected to the association end attached to *SortalType*. Although it is true that instances of $SortalType_i$ must be associated with at least y_1 instances of $EndurantType_i$, it is not the case that instances of *SortalType* in general must satisfy the same constraint. For this reason, the cardinality constraint y_1 in the original pattern is relaxed to an optional cardinality constraint (i.e., $0..y_n$).

Finally, suppose that in the generalization set GS we have one or more occurrences of a a sortal type $SortalType_k$ connected by datatype relations D_i to respective enumerations DT_k. In this case, we replace this pattern by a structure in which all elements of each of these DT_k are included as elements of the newly created enumeration datatype GS aforementioned. The $SortalType_k$ itself is not include as an item in the enumeration GS.

For example, in the model of fig. 4, we have three occurrences of this pattern. By applying R4 to that model, we obtain the model of fig. 5. So, for example, in fig. 4, we have the phase partition 'developmental phase' connecting the phases 'Child' ($SortalType_j1$),'Teenager' ($SortalType_j2$) and 'Adult' ($SortalType_i$) to the sortal (another phase) 'Living Person' (*SortalType*). As one can observe, in the model of fig. 5 (see R4* in the lower part of the figure), this configuration is replaced by a structure in which we have, connected to the common supertype 'Living Person', an enumeration named 'Development Phase' that contains as members the values 'Child', 'Teenager' and 'Adult'. Moreover, since the phase 'Adult' ($SortalType_i$) was originally connected via a 'rents' relation ($relation_i$) to the type 'Car' ($EndurantType_i$) then, in the replacing structure, the relation 'rents' is now connected to the common supertype 'Living Person' (with the appropriate relaxing of cardinality constraints). As one can observe, the *rolename* (in this case, 'Personal Customer') connected to the abstracted phase 'Adult' is now represented in the association end of 'rents' connected to 'Living Person'.[8]

In fig. 5, we also see another matching graph for the application of R4. In this case, we can apply once more R4 over the 'life status' generalization set (phase partition). The result of this is depicted in fig. 6. As one can observe in this figure, a new generalization set termed 'Life Status' is produced including items representing the phase 'Deceased Person' but also all the items that are members of the enumeration 'Developmental Phase'. Following the aforementioned rules, the latter enumeration is absorbed in the former. Moreover, following the same rules, the phase 'Living Person' is not included in this latter enumeration. Finally, since 'Living Person' participates as a 'Personal Customer' in the relation 'rents' with 'Car', in the fig. 6, this relation (with its corresponding *rolenames* and relaxed cardinality constraints) is move to the kind 'Person' (the direct supertype of 'Living Person').

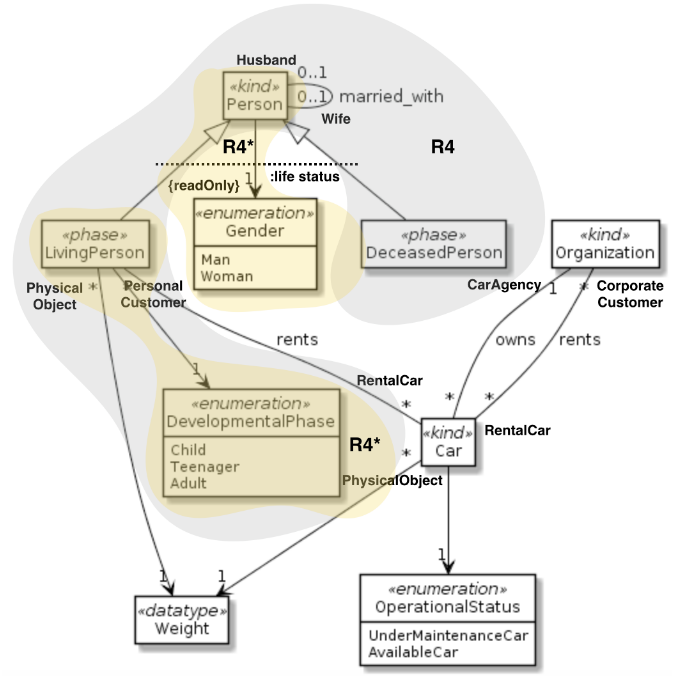


Fig. 5. Model resulting from the application of R4 on the model in fig. 4.

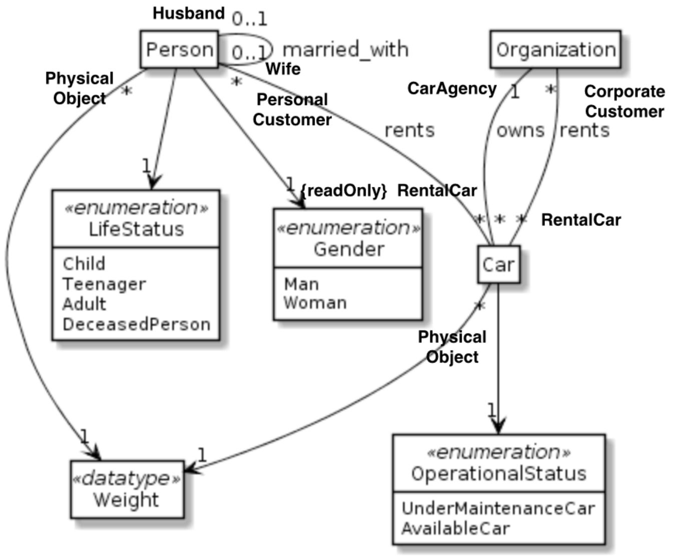


Fig. 6. Model resulting from the application of R4 on the model in fig. 5.

The model of fig. 6 is the final result of applying our

[8] After all, it is as personal customers that living people participate in car renting relations.

ruleset R1-R4 to the original model of fig. 1[9]. In such a final model, besides enumeration and datatypes, all remaining classes represent kinds and, hence, the annotations with the stereotype «kind» can be omitted. Notice that the union of these three kinds amounts to the objects that are really considered to exist in this universe of discourse. In other words, 'Person', 'Organization' and 'Car' tessellate the universe circumscribed by our conceptualization of the domain. As such, the final model can be said to represent the essence of this conceptualization. In fact, we argue that this model does capture the gist of the semantics of this domain while (in this case) achieving a compression rate of more than 70% in relation to the original model of fig.1[10].

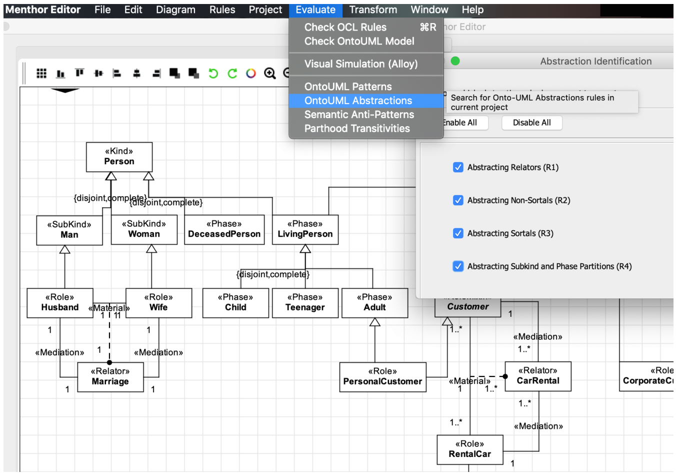
V. Computational Support and Evaluation

In this section, we describe a tool for Ontology-Based Model Abstraction implemented according to the graph-rewriting rules proposed in section IV. This tool was built as a plug-in for the Menthor Editor [20], an open-source ontology-driven conceptual modeling platform which incorporates the theories of the Unified Foundational Ontology (UFO). The tool supports modeling, verification, validation, and implementation of OntoUML models.

In this plug-in, by employing the explicitly defined MOF metamodel on which this editor is based, we have manage to implement a set of transformations that automatically generate different abstractions of OntoUML models following our proposed rule set. For instance, as one can observe in fig. 7, the tool can generate model abstractions corresponding to steps of the abstraction process, i.e., models resulting from the application of R1 (equivalent to fig.2), R2 (equivalent to fig.3, R3 (equivalent to fig.4) and R4 (equivalent to fig.5 and fig.6).

We have tested the tool against a number of models from the OntoUML model repository. These models vary in terms of size and setting in which they were produced (i.e., whether produced in an industrial or governmental setting, or in an academic environment). Moreover, they cover a variety of domains ranging from Biodiversity (OpenBio) and Heart Electrophysiology (ECG) to Optical Networks (G.805) and Governamental Organizations (MPOG). For a discussion about the OntoUML model repository and these particular models, the reader should refer to [22].

Table II reports on the result of these tests. The table includes a column reporting on the size of the original models in terms of the total sum of modeling elements. It also contains the compression rates for each of these models achieved by applying the proposed abstraction rule set. As one can observe, the latter range from 47,22% to 85,48%. Moreover, we also recorded the execution time for the largest of these models, namely, OntoBio (with 274 model elements) and for an even larger syntactically created model (with 777 model elements). In a MacBook Intel Core i5 2.3GHz with 4GB of RAM, the entire rules set was executed for these models in 3.061 seconds and 19.11 seconds, respectively.



Search for Onto-UML Abstractions rules in current project

☑ Abstracting Relators (R1)

☑ Abstracting Non-Sortals (R2)

☑ Abstracting Sortals (R3)

☑ Abstracting Subkind and Phase Partitions (R4)

Fig. 7. Screenshot of the implementation at the rule selection.

TABLE II
Testing the plug-in on a diverse set of OntoUML models

Model	#Modeling Elements (Original)	Compression Rate of the Abstracted Model
Cloud Vulnerability	83	72,28%
ECG	49	60%
G.805	139	85,48%
MPOG	15	51,35%
Normative Acts	64	62,14%
OpenBio	274	75,56%
Open Provenance	34	47,22%
OpenFlow	20	70,83%
PAS 77	76	61,39%
Software Requirements	23	59%

VI. Comparison to Related Work

The results presented in this paper complement the research in [9] as part of a general program of Ontology-Based Complexity Management of Large and Complex Conceptual Models. Whilst that work addresses the problem of model clustering by viewpoint extraction, here we focus on model abstraction, i.e., on a particular type of model summarization. In that sense, the approaches in the literature that are more

[9] In UML, *navigable association ends* are semantically equivalent to attributes. For this reason, this model is equivalent to a model in which the navigable associations connecting the types (kinds) Person, Car and Organization to the respective datatypes and enumerations are represented as attributes of these kinds. The alternative model consists of these three kinds occupying the center of the diagram while datatypes and enumerations can be left as secondary notational elements represented in the border of the diagram. This alternative visualization can, in principle, improve even further the presentation of the abstracted model. However, it is important to highlight that this move does not constitute a transformation, given that the models are equivalent. We refrain from further entertaining this possibility here.

[10] We calculate this compression rate by comparing the sums of all classes and relations present in the original model with the analogous sum in the abstracted one.

similar to the one presented here are the ones of Lozano et al. [15], [16] and Egyed [6], [7].

As previously discussed, the method of Lozano et al. only deals with viewpoint selection, thus, not providing any mechanism for abstraction. Moreover, the selection of views is largely dependent on the selection of focal concepts provided by the user. In particular, given the large degree of propagation of their rules throughout the model, the resulting views themselves might end up having a level of complexity that is similar to the original model. For instance, if for the model in fig.1 the user selects 'Car Rental' as a focal type, the resulting summarized model will contain all elements of the original model with the exceptions of the types 'Physical Object' and 'Ownership' (as well as the mediation and derived relation around it), and the datatype 'Weight'.

The approach of Egyed does provide both viewpoint selection and abstraction, i.e., the produced viewpoints are abstractions of the original fragments of the model. In particular, they are abstractions hiding elements in the paths connecting focal elements of the original model selected by the user. Like in the previous approach, the results produced by Egyed's method are partial models. In contrast, our method supports views that cover the entire scope of the underlying conceptualization. As such, our abstracted models can support not only the activities that are normally supported by model summarization (e.g., model understanding, consistency checking[11]and reverse engineering [7]) but also activities such as high-level appraisal of conceptual models (e.g., domain ontologies) in large model repositories. For example, as discussed by [23], an approach that affords 'glancing' over complex models in large repositories can support users in making well-informed decision about which model should be more suitable for reuse.

Finally, regarding the abstraction of conceptual models, we believe our approach offers two additional benefits when compared to the one proposed by Egyed: (i) simplicity of the abstraction rule set; (ii) determinism in producing the abstracted model.

Regarding (i), our proposal is constituted by 4 graph-rewriting rules. In contrast, in [7], Egyed proposes a set of 121 patterns, 92 of which are abstractions-generating rules. The main reason why we manage to rely on such a reduced set of rules is by leveraging on the highly-structured syntax of OntoUML models. Due to the ontological foundations behind the language, its modeling elements combine in very specific ways forming *modeling patterns*.[12]. In constrast, because of its ontological neutrality [12], UML elements can be combined in large variety of ways.

Also due to these ontological foundations, our approach can rely on well-demarcated differences between the multiple types of types (e.g., kinds, phases, roles, etc.) that can occur in OntoUML models, as well as on their different levels of ontological significance. This feature of the language has been systematically exploited, for example, in the rationale defining the order of application of our proposed rules R1-R4 (see section IV). In Egyed's approach, instead, there is no predefined order constraining the application of his rules. As a consequence, the user must reason with a multitude of possible sequences of applications, each of which can create rather different results. For instance, if we apply his approach to the model of fig. 1 and select as focus types both 'Person' and 'Car' we can produce both a model consistuted by 'Person', 'Personal Customer' and 'Car' but also an alternative model constituted by Person', 'Car' and 'Rental Car', depending on the selection of rules to be applied and the order in which they are applied.[13] In the case of large conceptual models, the number of alternative strategies for the application of these 92 rules explodes. Besides the inherent complexity, this introduces a second drawback in the approach, namely, the non-determinism of the produced models (point ii above).

The resulting viewpoints produced by Egyed's approach depend on a number of factors that are extrinsic to the model itself. As previously discussed, these factors include: the particular selection of focal types provided by the user; the particular selection of rules and their ordering in the process. A third type of factor is what Egyed calls 'model ambiguities' and these are related to the defesability of inferences produced by some of his rules. In contrast, our method always produces deterministic results and, for this reason, it can be fully automated. As observed by Egyed himself [7]: "given the large and complex nature of models, semi-automated abstraction can become very costly...[and it is not] computing time but human-intervention [that] constitutes key complexities in dealing with these large models."

VII. Final Considerations

In this paper, we propose an approach for model abstraction of conceptual models based on the ontologically well-founded semantics of the modeling language OntoUML. The main contribution of the work is to present a set of graph-rewriting rules and their definitions. These rules can automatically produce abstracted global versions of complex conceptual models, while simultaneously preserving the essential concepts of the model and their semantic relevance. By leveraging on the ontological semantics of OntoUML, the approach produces a version of the original model that is reduced to a small number of elements, in essence, the (ontologically fundamental and cognitively salient) *Kinds* of things that exist in that domain.

[11] For example, OntoUML is supported by an approach of model validation and anti-pattern detection by visual simulation [2], [22]. Two limitations of that approach are: the computational complexity of the simulation process - which relies on constraint satisfaction over a SAT solver; the cognitive tractability of the model instances generated by these simulations. We believe that the approach proposed here can be used to address both of these problems.

[12] In fact, as formally demonstrated in [33], OntoUML is a pattern language and its modeling primitives are actually design patterns representing the multiple micro-theories of the UFO foundational ontology

[13] Notice that, for example, that by selecting 'Living Person' and 'Car' as focus types, one can even produce a view including only as types 'Living Person','Personal Customer' and 'Car', i.e., a model without an identity-providing *kind* for the instances of 'Person'. Such a view would be problematic from an ontological point of view [12].

In an extension of this paper, we shall formally implement a specification of this rule set using the approach employed in [33] (a Single-Pushout Graph Transformation system and the GROOVE tool). The idea is to use an automated tool support to prove the consistency and completeness of the rule set. Moreover, we intend to extend this ruleset to address additional OntoUML constructs (e.g., events, part-whole relations, multi-level modeling structures). Although the ones dealt with here amount to the most used constructs in the language,[14] in certain domains, specific kinds of constructs can play a fundamental role (e.g., partonomies in biomedical applications, events in historical models, multi-level modeling in product design).

As a second contribution, the paper also presents a fully implemented plug-in tool for a Model-Based OntoUML Editor that automates these rules. Despite the encouraging results of the tests performed with this plug-in, we intend to subject it to a more comprehensive and systematic series of tests. Moreover, despite the empirical demonstration of performance scalability with these tests, we shall provide a formal proof of the complexity of the algorithm implemented therein. As an ultimate goal of this project, we intend to develop a version of the tool that supports not only the compression of models but also their 'unfolding' back to its original form (i.e., round-trip operations of model abstraction and refinement). The idea is to fully implement a notion of *Ontological Zoom* that would allow users to interact with models and their parts in different levels of abstraction (each level corresponding to the result of a transformation using each of the rules proposed here).

In order to properly evaluate the cognitive effectiveness of these contributions, we are already in the process of designing a series of empirical studies. The core focus concerns speed and accuracy of obtaining information from the model, as well as balancing this (hypothesized) improvement with the model's information loss.

References

[1] Akoka, J., Comyn-Wattiau, I.: Entity-relationship and object-oriented model automatic clustering. Data & Knowledge Engineering **20**(2), 87–117 (1996)

[2] Benevides, A.B., Guizzardi, G., Braga, B.F.B., Almeida, J.P.A.: Validating modal aspects of ontouml conceptual models using automatically generated visual world structures. J. UCS **16**(20), 2904–2933 (2010)

[3] Bodart, F., Patel, A., Sim, M., Weber, R.: Should optional properties be used in conceptual modelling? A theory and three empirical tests. Information Systems Research **12**(4), 384–405 (2001)

[4] Brasileiro et al.: Applying a multi-level modeling theory to assess taxonomic hierarchies in wikidata. In: 25th International Conference on World Wide Web (WWW). pp. 975–980 (2016)

[5] Castano, S., De Antonellis, V., Fugini, M.G., Pernici, B.: Conceptual schema analysis: Techniques and applications. ACM Trans. Database Syst. **23**(3), 286–333 (Sep 1998)

[6] Egyed, A.: Semantic abstraction rules for class diagrams. In: Automated Software Engineering, 2000. Proceedings ASE 2000. The Fifteenth IEEE International Conference on. pp. 301–304. IEEE (2000)

[7] Egyed, A.: Automated abstraction of class diagrams. ACM Transactions on Software Engineering and Methodology **11**(4), 449–491 (2002)

[8] Egyed, A.: Compositional and relational reasoning during class abstraction. In: International Conference on the Unified Modeling Language. pp. 121–137. Springer (2003)

[9] Figueiredo, G., Duchardt, A., Hedblom, M.M., Guizzardi, G.: Breaking into pieces: An ontological approach to conceptual model complexity management. In: IEEE 12th International Conference on Research Challenges in Information Science. pp. 1–10. IEEE (2018)

[10] Francalanci, C., Pernici, B.: Abstraction levels for entity-relationship schemas. In: International Conference on Conceptual Modeling. pp. 456–473. Springer (1994)

[11] Guarino, N., Guizzardi, G.: 'We need to discuss the relationship': Revisiting relationships as modeling constructs. In: Intl. Conf. on Advanced Information Systems Engineering. pp. 279–294. Springer (2015)

[12] Guizzardi, G.: Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology (2005)

[13] Guizzardi, G.: Ontological meta-properties of derived object types. In: 24th International Conference on Advanced Information Systems Engineering (CAiSE). pp. 318–333 (2012)

[14] Guizzardi, G., Wagner, G., Almeida, J.P.A., Guizzardi, R.S.S.: Towards ontological foundations for conceptual modeling: The Unified Foundational Ontology (UFO) story. Applied Ontology **10**(3-4), 259–271 (2015)

[15] Lozano, J., Carbonera, J., Abel, M., Pimenta, M.: Ontology view extraction: an approach based on ontological meta-properties. In: Tools with Artificial Intelligence (ICTAI), 2014 IEEE 26th International Conference on. pp. 122–129. IEEE (2014)

[16] Lozano, J., Carbonera, J.L., Abel, M.: A novel approach for extracting well-founded ontology views. In: JOWO@ IJCAI (2015)

[17] Macnamara, J.T., Macnamara, J., Reyes, G.E.: The logical foundations of cognition. No. 4 in Vancouver Studies in Cognitive Science, Oxford University Press on Demand (1994)

[18] Moody, D.L., Flitman, A.: A methodology for clustering entity relationship models: a human information processing approach. In: International Conference on Conceptual Modeling. pp. 114–130. Springer (1999)

[19] Moody, D.L., Flitman, A.R.: A decomposition method for entity relationship models: A systems theoretic approach. In: ICSTM (2000)

[20] Moreira, J.L.R., Sales, T.P., Guerson, J., Braga, B.F.B., Brasileiro, F., Sobral, V.: Menthor editor: An ontology-driven conceptual modeling platform. In: JOWO@ FOIS (2016)

[21] Recker, J., Rosemann, M., Green, P.F., Indulska, M.: Do ontological deficiencies in modeling grammars matter? MIS Quarterly **35**(1), 57–79 (2011)

[22] Sales, T.P., Guizzardi, G.: Ontological anti-patterns: Empirically uncovered error-prone structures in ontology-driven conceptual models. Data & Knowledge Engineering **99**, 72–104 (2015)

[23] Tartir, S., Arpinar, I.B.: Ontology evaluation and ranking using ontoqa. In: Proceedings of the International Conference on Semantic Computing. pp. 185–192. ICSC '07, IEEE (2007)

[24] Teixeira, M.G.: An ontology-based process for domain-specific visual language design. Federal University of Espirito Santo, Brazil/Ghent University, Belgium (Double Degree) (2016)

[25] Tzitzikas, Y., Hainaut, J.L.: How to tame a very large er diagram (using link analysis and force-directed drawing algorithms). In: International Conference on Conceptual Modeling. pp. 144–159. Springer (2005)

[26] Tzitzikas, Y., Kotzinos, D., Theoharis, Y.: On ranking rdf schema elements (and its application in visualization). J. UCS **13**(12), 1854–1880 (2007)

[27] Verdonck, M., Gailly, F.: Insights on the use and application of ontology and conceptual modeling languages in ontology-driven conceptual modeling. In: International Conference on Conceptual Modeling. pp. 83–97. Springer (2016)

[28] Verdonck et al.: Comparing traditional conceptual modeling with ontology-driven conceptual modeling: An empirical study. Information Systems (2018)

[29] Villegas Niño, A.: A Filtering Engine for Large Conceptual Schemas. PhD Thesis. Universitat Politècnica de Catalunya (2013)

[30] Wieringa, R., de Jonge, W., Spruit, P.: Using dynamic classes and role classes to model object migration. TAPOS **1**(1), 61–83 (1995)

[31] Xu, F.: From lot's wife to a pillar of salt: Evidence that physical object is a sortal concept. Mind & Language **12**(3-4), 365–392 (1997)

[32] Xu, F., Carey, S.: Infants metaphysics: The case of numerical identity. Cognitive psychology **30**(2), 111–153 (1996)

[33] Zambon, E., Guizzardi, G.: Formal definition of a general ontology pattern language using a graph grammar. In: Federated Conference on Computer Science and Information Systems. pp. 1–10 (2017)

[14] This is demonstrated, for example in [24] by an analysis of an OntoUML model repository with circa 56 models in a variety of domains.