Abstracting Ontology-Driven Conceptual Models: Objects, Aspects, Events, and Their Parts

Elena Romanenko[1(✉)], Diego Calvanese[1,2], and Giancarlo Guizzardi[1,3]

[1] Free University of Bozen-Bolzano, 39100 Bolzano, Italy
{eromanenko,giancarlo.guizzardi}@unibz.it, calvanese@inf.unibz.it
[2] Umeå University, 90187 Umeå, Sweden
[3] University of Twente, 7500 Enschede, The Netherlands

Abstract. Ontology-driven conceptual models are widely used to capture information about complex and critical domains. Therefore, it is essential for these models to be comprehensible and cognitively tractable. Over the years, different techniques for complexity management in conceptual models have been suggested. Among these, a prominent strategy is model abstraction. This work extends an existing strategy for model abstraction of OntoUML models that proposes a set of graph-rewriting rules leveraging on the ontological semantics of that language. That original work, however, only addresses a set of the ontological notions covered in that language. We review and extend that rule set to cover more generally types of objects, aspects, events, and their parts.

Keywords: Conceptual model abstraction · Complexity management of conceptual models · OntoUML

1 Introduction

The term conceptual model (CM) is heavily overloaded and covers a wide range of models, e.g., Entity-Relationship diagrams and Business Process Models. These models are used to capture information about complex and critical domains, and "play a fundamental role in different types of critical semantic interoperability tasks" [14].

Conceptual modeling is the activity of "representing aspects of the physical and social world for the purpose of understanding and communication [...] among human users" [19]. However, when ontological theories, coming from areas such as formal ontology, cognitive science, or philosophical logics, are utilized for improving the development of CMs, it is common to speak of ontology-driven conceptual modeling (see [7,22]).

In critical and complex scenarios, the number of concepts and axioms of a CM can grow significantly, leading to situations where "it is important that conceptual models are cognitively tractable" [6]. It is known that human working memory capacity in processing visual information is limited [15], and "displaying

R. Guizzardi et al. (Eds.): RCIS 2022, LNBIP 446, pp. 372–388, 2022.
https://doi.org/10.1007/978-3-031-05760-1_22

a large amount of data in a single node-link diagram can be visually overwhelming and confusing" [15]. Thus, one of the most challenging aims is "to understand, comprehend, and work with very large conceptual schemas" [23].

Due to the above-mentioned reasons, complexity management of large conceptual models has been an area of intensive research. Recently published methods can be grouped into the following categories: clustering methods, relevance methods, and summarization methods [23, p.54]. The first group covers methods in which elements of the CM are divided into groups (clusters). Relevance methods rank CM elements into ordered lists according to their value, while summarization methods produce a reduced version of the original CM.

The work presented in this paper belongs to the group of summarization techniques, more specifically, to the abstraction techniques. According to Egyed [5], model abstraction is "a process that transforms lower-level elements into higher-level elements containing fewer details on a larger granularity". The main idea is to provide the user with a bird's-eye view of the model by filtering out some details. Thus, such methods by definition provide lossy transformations.

Egyed also suggested an interesting approach for model abstraction [5]. The proposed abstraction algorithm performs syntactic matching of abstraction rules on the model, and the matched pattern is replaced by the result pattern of that rule. The author claims that every application of a rule simplifies a given model. However, the suggested set of rules is complicated, and it consists of 121 patterns, 92 of which are abstraction-generating rules.

Since most of the methods for CM summarization are based on classic modeling notations (UML, ER) [23, p.44], they rely on syntactic properties of the model, such as closeness or different types of distances between model elements (see [1]). What is interesting here is that abstraction techniques even for languages with ontological semantics sometimes are based mainly on topological properties of the graphs, see [16,20].

More recently, a number of approaches for complexity management have been proposed for Ontology-Driven CM languages—most notably OntoUML [9]—by leveraging on the richer ontological semantics offered by these languages. These include [6,12,18]. The latter deals exactly with the topic of model abstraction and proposes a set of graph-rewriting rules for abstracting OntoUML patterns. However, it targets only a subset of the ontological notions present in OntoUML. For example, it did not cover any other relation but specialization, while recently OntoUML stereotypes for relations have been revised and extended (see [7]). Moreover, the technique is focused on the category of objects, leaving out categories of aspects and events and, hence, also the relations between events and endurants (objects and aspects). In this paper, we revisit and extend that work by proposing new abstraction graph-rewriting rules that more generally address types of objects, aspects, events, and their parts.

The remainder of the paper is organized as follows: Sect. 2 presents our baseline and background; Sect. 3 introduces our new abstraction rule set, which is combined with the original rule set in an unified algorithm in Sect. 4; Sect. 5 elaborates on final considerations and future work.

2 Background

2.1 A Brief Introduction to UFO and OntoUML

Unified Foundational Ontology (UFO) [9,11] is a well-grounded foundational ontology based on contributions from Formal Ontology in Philosophy, Philosophical Logic, Cognitive Psychology, and Linguistics. A quite recent study [21] has shown that UFO is the second-most utilised foundational ontology applied in conceptual modeling, and also the one with the fastest adoption rate.

OntoUML is an ontology-driven conceptual modeling language that extends UML class diagrams by defining a set of stereotypes that reflect UFO ontological distinctions into language constructs. These stereotypes extend UML's metamodel via a profile mechanism, and allow users to decorate diagram's elements. Thus, classes and associations that are decorated with OntoUML stereotypes bring precise (real-world) semantics grounded in the underlying UFO ontology. Additionally, a number of semantically motivated syntactic constraints govern OntoUML models, making them compliant with UFO [9]. Verdonck and Gailly also have shown that OntoUML is among the most used languages in ontology-driven conceptual modeling [21]. In the remainder of this section, we briefly describe a selected subset of the ontological distinctions in UFO, and how they are represented by means of OntoUML. For an in-depth discussion, philosophical justification, and formal characterization we refer to [9,13].

A first key distinction is made between *Endurants*, i.e., entities that exist in time while keeping their identity (e.g., `John`, `Mary` and their `Marriage`), and *Perdurants*, i.e., exist that occur in time, also generally called *Events* (e.g., `John and Mary's Wedding ceremony`). Endurants instantiate *Endurant Types*, which depending on their modal properties can be classified into *Ultimate Sortals*, *Subkinds*, *Roles*, *Phases*, *Categories*, *Role Mixins*, *Phase Mixins*, and *Mixins*. Perdurants instantiate *Perdurant Types*.

Ultimate Sortals are types that capture the essential properties ascribed to their instances. The term is a synonym to what is called a (natural) kind in the literature. OntoUML reserves the stereotype *Kind* for *Object* kinds (e.g., `Car`, `Person`, `Planet`) but also allows *Relator* kinds (e.g., `Marriage`, `Employment`, `Enrollment`), *Quality* kinds (e.g., `Color`, `Weight`), and *Mode* kinds (e.g., `That conductor's electrical conductivity`, `Mary's ability to speak English`, `John's Dengue Fever`). *Objects* are endurants that can exist independently, while *Relators*, *Modes* and *Qualities* (generally called *Aspects*) can only exist parasitic to other entities (existential dependence): *Relators* are the truthmakers of relational propositions, e.g., a `Marriage` is a complex relator composed of mutual commitments and claims; *Qualities* are entities that are existentially depend on their bearers and can be directly measured; *Modes*, on the contrary, represents complex intrinsic aspects that can bear their own properties.

Subkinds are subtypes specializing ultimate sortals, like `Man` and `Woman` for the kind `Person`, or `Temporary Employment` for the relator kind `Employment`. Ultimate sortals are static (or *Rigid*) types in the sense that they classify their instances necessarily (in the modal sense). For this reason, they are called *Rigid*

Sortals. *Phases* and *Roles* represent contingent (accidental, dynamic) subtypes of rigid sortals. They are hence called *Anti-Rigid Sortals*. The former class represents specializations according to intrinsic properties of instances, like being a `Child` or an `Adult` for `Person`, while the latter is the way for endurants to participate in relations, e.g., play the role of `Wife` in a `Marriage`. Rigid and anti-rigid sortals constitute the class of *Sortal Types*.

Non-Sortal Types capture properties that are common to instances of different kinds (ultimate sortals). These can be essential properties, in which case these are called *Categories*; non-essential properties, in which case these are called *Mixins*; contingent properties cross-cutting several types, in which case these are called *Role Mixins* and *Phase Mixins*. An example of a role mixin is the type `Customer` that describes properties that apply to both individuals of the kind `Person` and individuals of the kind `Organization`.

Events bear different relations to endurants. On the one hand, they can create, change, or terminate them. On the other hand, events are manifestations of aspects, like when the `unfolding of John's Dengue Fever` (an event) manifests a `particular set of pathological conditions in his body` (a mode). In this case, by being the bearer of this mode, John *participates* in that corresponding dengue fever unfolding event. *Event Types* can also specialize other event types, thus, forming their own taxonomies (e.g., `Cardiovascular Surgery` is a subtype of `Surgery` event).

2.2 Ontology-Based Model Abstraction

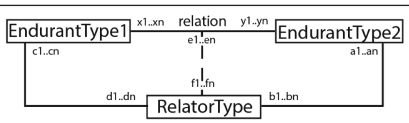
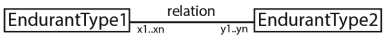
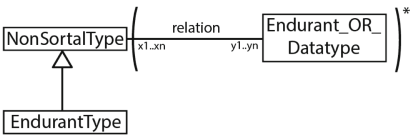
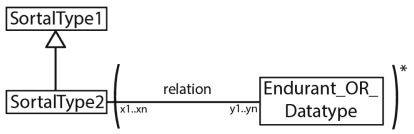
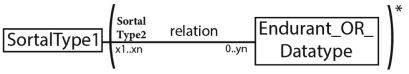
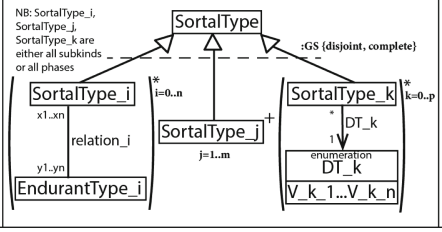
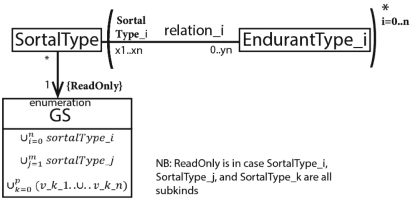
The approach presented in [12] is made possible by the ontological semantics of OntoUML. The approach proposed an algorithm that is based on four graph-rewriting rules, `R1`–`R4` (see Table 1, reproduced from [12]). Following these rules, fragments of the original model are abstracted into reduced counterparts, while maintaining essential information. The rationale behind these rules is that the focus should be on the representation of *Object* kinds and relations between them, so that: *(1) Relators* are to be omitted; *(2)* properties of *Non-Sortals* (e.g., *Categories*) should be moved downwards to the level of *Sortals*; and *(3)* properties of *Sortals* that are not *Object* kinds, such as *Subkinds* and *Roles*, should be pushed upwards to the level of kinds. Given this rationale, the rules are meant to be applied in a specific order.

As previously discussed, there is the need to revise and extend this approach. First, it only covers subtyping relations. Second, it neglects both taxonomies of *Aspect Types* and *Event Types*. Given that latter are not addressed, so are also the relations between *Events* and *Endurants*.

3 Abstracting Objects, Aspects, Events, and Their Parts

In the next three subsections, we introduce our new rules for abstracting types of *Objects*, *Aspects*, and *Events*, respectively. Each subsection describes first parthood relationships and then other ontological relations involving these types of

Table 1. Original graph-rewriting rules for OntoUML model abstraction [12].

Rule	Matching Graph	Replacing Graph
R1		
R2		
R3		
R4		

entities, the summary of which is given in Fig. 1[1]. The section ends with a list of graph-rewriting rules.

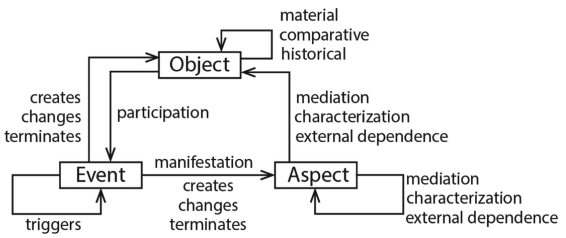


Fig. 1. *Types* and stereotyped relations between them.

[1] A formal definition of these stereotyped relations is given in [7] and in [2] for those relations that include *Events*.

3.1 Abstracting Objects

In the previous version of the algorithm [12], parthood relations were not considered. Despite the apparent simplicity, mereological relationships, also known as part-whole or part-of relationships, come in different forms, which behave differently in practice. An interesting research question is whether this diversity leads to different rules when abstracting such relationships and, if yes, how these rules could be formalized.

There are many mereological systems described in logic, philosophy and related fields. These systems differ depending on what axioms are included. However, the "minimal characterization of parthood relation", P, is provided by three axioms [4, Ch. 2] amounting to is termed *Minimal Mereology*:

$$\forall x P(x,x) \qquad \text{(Reflexivity)}$$
$$\forall x \forall y((P(x,y) \wedge P(y,x)) \rightarrow x = y) \qquad \text{(Antisymmetry)}$$
$$\forall x \forall y \forall z((P(x,y) \wedge P(y,z)) \rightarrow P(x,z)) \qquad \text{(Transitivity)}$$

However, in real-world scenarios transitivity does not always hold (see [10]). As an example, we model the case where hearts of football players can fail during a game, thus, leading to a surgery[2] (see Fig. 2)[3]. When a player as part of a team scores a goal, the whole team scores a goal. In contrast, although a football player (as a person) has a heart, it seems rather odd to speak about the heart (being part) of a football team (at least in the biological sense).

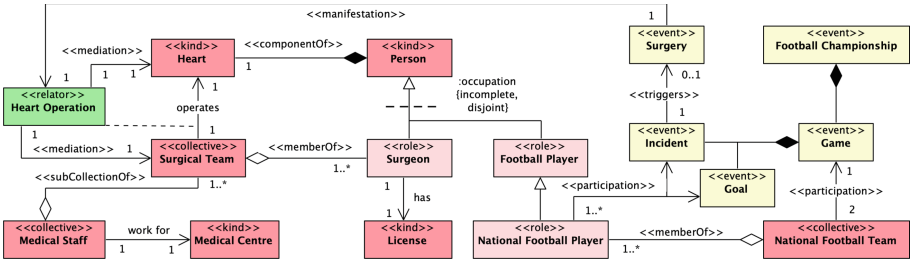


Fig. 2. Examples of different parthood relations.

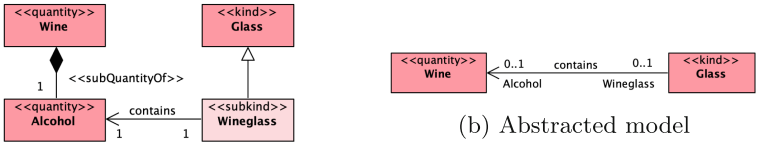
[2] Unfortunately, incidents like this from time to time happen in real life, e.g., https://www.webmd.com/heart-disease/news/20210614/danish-soccer-player-suffered-cardiac-arrest-during-euro-match.

[3] Hereinafter, the default cardinality constraints '$*$' and '0..$*$' are not shown in the models, as well as '1' on the side of diamonds in parthood relations. For visual economy, when we have more than one part type connected to the same whole type, we join these different parthood relations in the same diamond-head (on the end connected to the whole). Parthood, nonetheless, is still defined as a binary relation.

In computer science, UML standard distinguishes between an aggregation and a composite aggregation, where the latter is a strong form that "requires a *part object* to be included in *at most one composite object* at a time. If a composite object is deleted, all of its part instances that are objects are deleted with it." [3, p. 112] Also, "compositions may be linked in a directed acyclic graph with *transitive deletion* characteristics" [3]. This standard approach does not provide enough support for the modeller to distinguish different types of parthood relationships but addresses the problem of possible independence of parts.

As for OntoUML, different types of parthood relations between *Objects* and the problem of their transitivity were considered in [9,10]. According to the specification of UFO [9,11], there are three types of *Objects* that can participate in parthood relations: *Functional Complexes*, *Collectives*, and *Quantities*.

Quantities are connected to their parts with the *SubQuantityOf* relation, which is always transitive [9, p. 184] but quite rare in practice. Since the relationship between the portion and the whole is very close, these quantities coalesce their mass without additional attributing. And because of transitivity, a relation in which a portion serves as domain can be abstracted to the whole with the proper role. As an example, we can take `Alcohol` as a sub quantity of `Wine` contained in a `Wineglass`. After abstraction we obtain that `Wineglass`, or even `Glass` when abstracting further, contains `Wine` (see Fig. 3).



(a) Original model

(b) Abstracted model

Fig. 3. Abstracting *SubQuantityOf*.

A *Collective* can be part of another *Collective* via the *SubCollectionOf* relation, which is transitive [9, p. 186]. Thus, `Surgical Team` from Fig. 2 should receive all properties from `Medical Staff`, e.g., an employment contract with a `Medical Centre`. *Collective* also includes elements with the *MemberOf* relation, which on the contrary is intransitive [9, p. 185].

While formulating the abstraction rules for *Collectives*, we need to take into consideration the following remarks. First, members of the collection by definition of the relation have a uniform structure and are conceived as playing the same role in the collection, like a `Player` in a `Team`. If it is not the case, and we want, e.g., to distinguish between `Forward` and `Goalkeeper`, additional subcollections shall be introduced, each of which with uniform members. Second, "although member-collective is never transitive, a combination of member-collective and subcollective-collective is again *always transitive*" [10], i.e., in our example, a `Surgeon` as part of the collective also works for a `Medical Centre`.

Finally, "the member-collective relation necessarily causes the part to be seen as atomic in the context of the whole, hence, 'blocking' a possible transitive chain of part-whole relations" [9]. Thus, chains of *Collectives* could be abstracted to the most general collection together with the propagation of all relations in which these subcollections are domains. However, members of the collections must be kept because of their atomicity and relative independence. In the example from Fig. 2, this approach gives us `Medical Staff` that works for `Medical Centre`, operates `Heart` (as `Surgical Team`), and has `Surgeon`s as members.

Functional Complexes include their parts with the *ComponentOf* relation, which is not transitive in general [9, p. 183]. The simplest approach is to use these parts as attributes of the whole (see `heart` of `Person` in Fig. 4).

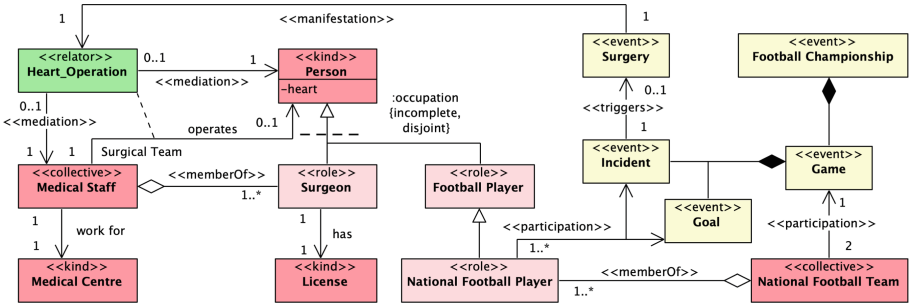


Fig. 4. Abstracting parthood relations of *Objects*.

However, there is no general rule for abstracting relations in which these parts serve as domains. In fact, the only relation that also holds for the whole object is `<<participation>>` in the *Event*, e.g., if `Heart` participated in a `Surgery`, a `Person` with that heart also participated in the same `Surgery`[4].

Abstracting from other relations that bind parts of *Objects* to other *Objects* is more challenging. As an example we can consider `Tail` as a component of `Dog`, which is used for `Greeting`. Abstracting these relations to the `Dog` '`used for`' `Greeting` is definitely wrong. Instead of removing such relations, we suggest to rename them using the pattern '*Object's part RelationName*'. In the given example, that would result in `Dog` connected to `Greeting` via the relation '`Dog's tail used for`'. Similar examples can be given for the `<<comparative>>` and `<<historical>>` stereotypes.

It should be noted, that so far we did not discuss situations where part of the *Object* serves as range of the relation coming from *Aspect* or *Event*. These rules with proper substantiation are considered in the corresponding sections.

After abstracting from parthood relations, further abstractions of *Objects* can be made with the help of rules from the previous version of the algorithm [12], namely, Rules `R2`–`R4` can be applied.

[4] This is called the *Principle of Event Expansion* [17].

3.2 Abstracting Aspects

The first rule suggested in [12] for abstracting models was `R1`, the rule for eliminating *Relators*. We argue that this rule can be further extended towards other *Aspects* of relations. As an example, we use the model in Fig. 5.

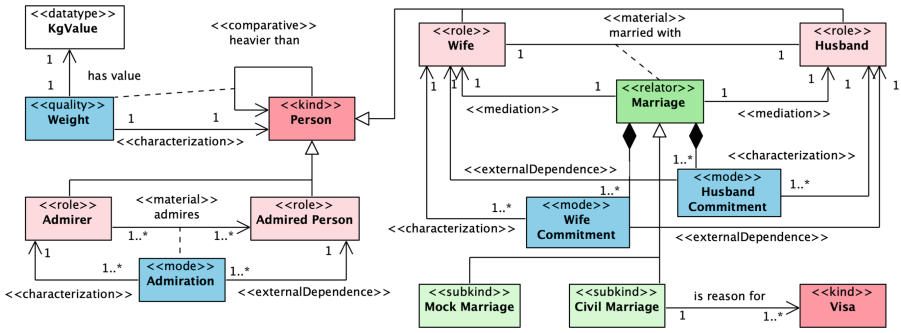


Fig. 5. Different *Aspects* of relations.

As previously discussed [8,9], *Relators* are the truthmakers of relational propositions, and their absence may lead to single-tuple/multiple-tuple cardinality ambiguity problem. Therefore, by applying `R1`, we have a *lossy transformation*.

What was omitted in [12] are the following features. First, *Relators* as *Endurants* could be embedded into a hierarchy and have *Subkinds* or *Phases* (an example can also be found in [8]). Second, *Relators* are complex objects in terms of how they are formalized as mereological sums of externally dependent *Modes* (for details see [7]). Third, *Relators* or their descendants may participate in other relations other than mediation relations with the relata. In the given example, `Civil Marriage` can serve as a reason for a spouse's `Visa`, while `Mock Marriage` cannot.

The last version of UFO distinguished *Relators*, *Qualities*, and *Modes* within the *Aspect Types* [11]. We argue that pattern for abstracting should be the same for all pre-described classes. Also, the above mentioned comments about the *Relators* can be applied to *Qualities* and *Modes* as well (see Fig. 7).

The first step when abstracting *Aspects* is addressing aspect parthood. All parts of an *Aspect* follow classical mereology axioms that were mentioned in Sec. 3.1. Also, because of transitivity, it is possible to move relations from a part of the *Aspect* to the whole. However, in most cases, like the one in Fig. 5 where `Wife Commitment` and `Husband Commitment` are parts of `Marriage`, a relation could already exist[5] and there is no need to create a new one.

[5] For details, why this is the case, see Table 1 in [7].

The second step is to abstract from aspect taxonomies by applying a modification of the original `R3` rule, since we are not interested in creating an enumeration like in `R4` even for disjoint and complete generalization sets because the transformation is lossy. The modification is the following. First, instead of applying `R3` to *Sortal Type* (meant there as Sortal Object Type), we define an analogous rule for *Sortal Aspect Type*. Second, we need to keep the available role that is lowest in the hierarchy. In the example from Fig. 5, that leads to `Marriage` (as `Civil Marriage`) being a reason for a `Visa`.

The last step consists of abstracting from the *Aspect Type* itself. However, it can participate in the relations, decorated with `<<externalDependence>>`, `<<mediation>>` and `<<characterization>>` stereotypes, representing different sorts of existential dependencies (for details see [7]). Also, if the *Aspect Type* participates in the relations as the domain, those relations should be moved to the corresponding *Object Types* (i.e., those that are ranges for `<<mediation>>` and `<<characterization>>` relations only) with the corresponding modification of the relation as the '*Object's AspectRole RelationName*'. The resulting model for our example is shown in Fig. 6.

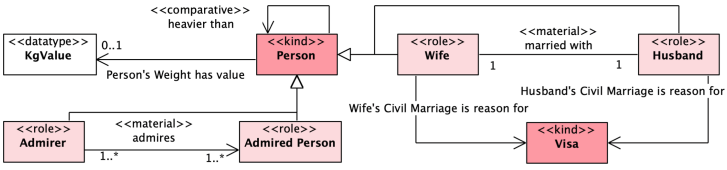


Fig. 6. Resulting model for abstracting *Aspects*.

The last rule gives us a reason to move `<<externalDependence>>` and `<<characterization>>` with `<<mediation>>` relations from part of an *Object* to the whole, even for intransitive parthood relations, but with some priority. So, `<<mediation>>` and `<<characterization>>` relations, being relations of *inherence* (i.e., a stronger for existential dependence), are more important to keep than `<<externalDependence>>`. Thus, in Fig. 4, `Heart Operation` mediates `Medical Staff` and `Person`.

Finally, *Aspects* can have relations with other *Aspects*. See, e.g., Fig. 7, where `Redirected Destination Intention` characterizes another *Mode* `Walk`. Because of that, the three above-mentioned steps, namely *(1)* abstracting from parthood, *(2)* abstracting from taxonomic relations, and *(3)* abstracting from the *Aspect Type* itself, should be applied iteratively. The result for our example is given in Fig. 8.

3.3 Abstracting Events

Event mereology is quite extensively described in [2]. Here we also take that *Event* may be composed of other *Events*. This parthood relation is the standard

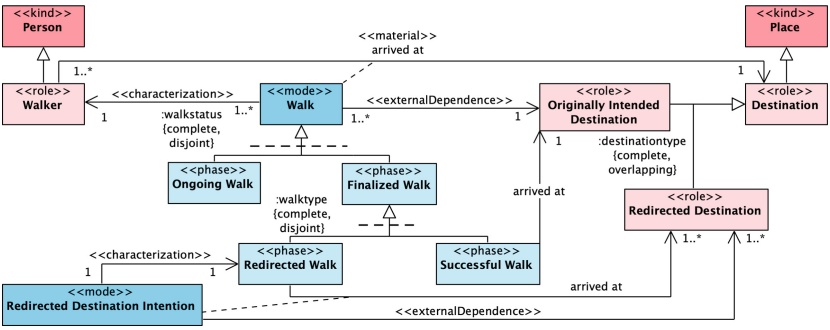


Fig. 7. Relations between *Aspects* (based on model from [11]).

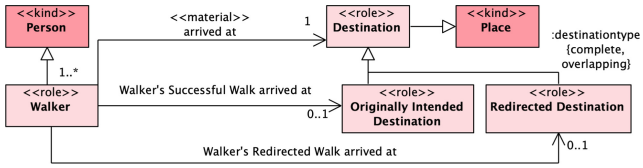


Fig. 8. Result of abstraction from *Aspects*.

one and obeys the three previously mentioned axioms. Thus, we can follow the same approach as for *Aspects* and abstract parts of the *Event* to the whole. Taking into account these arguments, in our example we can abstract the five *Events* in Fig. 4 to `Surgery` and `Football Championship` only.

The next aspect concerning the abstraction of *Events* is dealing with relations. Earlier we have mentioned that if part of an *Object* participates in an *Event*, the whole *Object* also inherits this relation. However, [2] mentioned that the opposite is also true, and when an *Object* participates in part of an *Event* it also participates in the whole *Event*. In the given example, if a `Football Team` participates in at least one `Game` of the `Football Championship`, it participates in the whole `Championship` (in the sense that Pelé played the 1962 World Cup by just playing a few games).

In some of the relations, *Events* can be associated with domains. First, we claim that a `<<triggers>>` relation, where both the domain and the range are *Events*, can be moved safely to more general *Events*.

Second, `<<creates>>` and `<<changes>>` relations from an *Event* to part of an *Object* or *Aspect* can also be abstracted to the whole type. However, the `<<terminates>>` relationship is trickier, because we can terminate the whole only if this part is an *essential* part[6]. In OntoUML notation, this is expressed as an `essential` tagged value decorating that parthood relation. Coming back to the example with `Person`, one can conclude that the termination of the `Brain`

[6] For the definition of essential and inseparable parthood, we refer to [9].

would result in the termination of the `Person` as a whole, but at the same time some other organs could be not strictly required.

Finally, *Events* are always manifestations of *Aspects* (or their aspect parts). Coming back to the example in Fig. 5, we can imagine the `Wedding` *Event*, which would explicitly manifest `Fiancée Commitments` and `Fiancé Commitments`. Abstracting from *Aspects* should lead to moving this manifestation to the proper *Relator*, in this case `Engagement`. As a consequence, we also have that the relata at hand participate in this `Wedding`, thus, `Fiancée` and `Fiancé` have to participate in their `Wedding`.

Events can also form taxonomies. E.g., in our first example, instead of `Surgery`, we can consider `Cardiovascular Surgery` as a *Subkind* of the more general `Surgery`. They can also be abstracted with the same rule suggested in the previous section, but now with *Event Type*. However, contrary to *Aspects*, *Events* are not completely abstracted at the end, so a modification of Rule `R4` is suggested in which all *Subkinds* of an *Event* form an enumeration.

Taking into account all these considerations, in our example, the model in Fig. 2 can be abstracted to the one in Fig. 9.

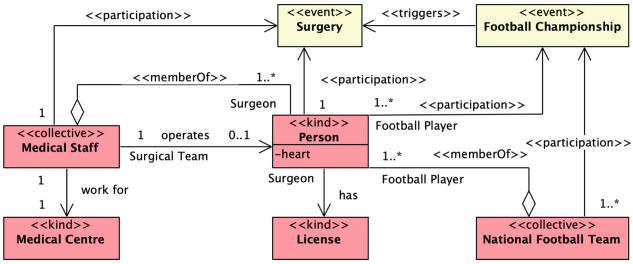


Fig. 9. After abstracting *Events*, *Aspects* and *Objects*.

3.4 Combining Abstraction Rules

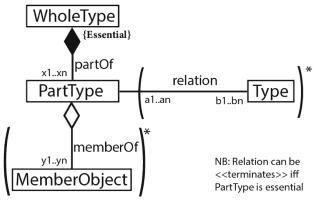
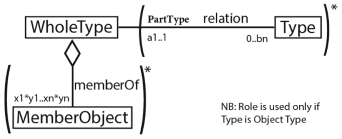
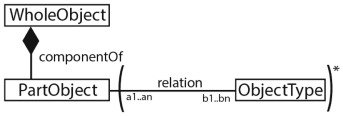
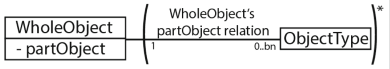
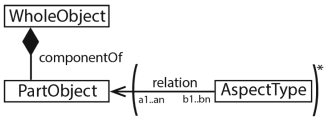
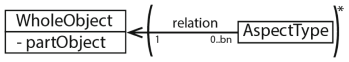
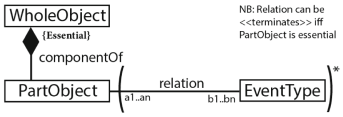
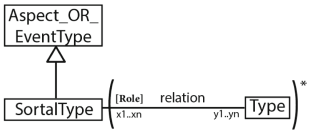
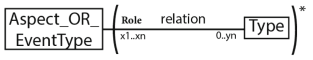
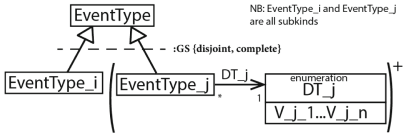
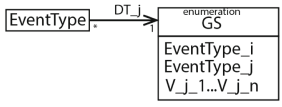
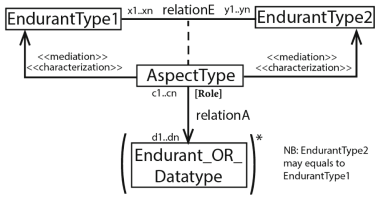
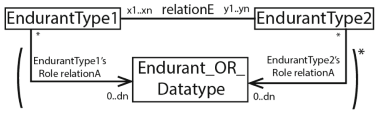
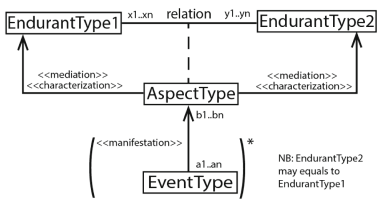
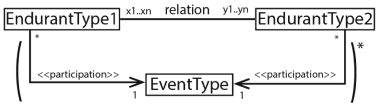
Table 2 provides a summary of all previously suggested abstraction rules.

The first set of rules, `P`, is responsible for abstracting from parthood relations. The first variant of the rule is applicable to parthood relations with transitive properties (called `partOf` in general) and it can be applied to *Objects*, *Aspects*, and *Events*. The only exception there is w.r.t. the relations is termination. If we suppose that `Alcohol` is an essential part of `Wine`, its exhalation would lead to the 'end' of `Wine`. Also, if there was already a relation between a whole and another type, there is no need to create a new one.

All other rules from this set are applied to *Object Types* only and may be applied together to the same type, i.e., they are not mutually exclusive, but could work as a supplement to each other. The difference between them is in the *Type* that has a relation with the part of the *Object* at hand.

The second set of rules, `H`, is responsible for abstracting from hierarchies. The previously defined Rules `R2`–`R4` are still applicable to *Objects*, but two new

Table 2. Graph-rewriting rules for OntoUML model abstraction

Rule	Matching Graph	Replacing Graph
P: Abstracting Parthood 1		
P: Abstracting Parthood 2		
P: Abstracting Parthood 3		
P: Abstracting Parthood 4		
H: Abstracting Hierarchy 1–3	Rules R2–R4	
H: Abstracting Hierarchy 4		
H: Abstracting Hierarchy 5		
A: Abstracting Aspects 1		
A: Abstracting Aspects 2		

rules were introduced. Rule H4 is applicable both to *Aspect Types* and to *Event Types*. In contrast to the rules from the first set, all relations with different role names should be kept. Rule H5 keeps an enumeration of all *Subkinds* of *Events* and can be used together with Rule H4.

The last set of rules, A, is responsible for abstracting *Aspect Types* and explicitly representing the participation of *Endurant* in *Events* that manifest these corresponding aspects. Again, these two rules could work in tandem.

4 Towards Ontology-Based Model Abstraction 2.0

As we previously mentioned, the work of [12] was focused on *Object Types* and restricted to subtyping relations between these types. Our revised proposal not only takes into account different *Types*, but also considers parthood relations and some other stereotyped relationships.

Considering the original proposal, the only rule that was completely reworked is Rule R1. Rules R2–R4 are still applied to *Object Types*, although, *Aspect Types* and *Event Types* were also addressed by new rules for abstracting their taxonomies.

What is more important, in contrast to the original proposal, there isn't a single strict order in which rules must be applied.

Thus, we here suggest two variants of rule's ordering that seem meaningful, namely, a *Parallel* (Listing 1) and an *Iterative* (Listing 2) versions of our new abstraction algorithm.

Listing 1. Parallel version of the abstraction algorithm

```
// abstract from all parthood relations:
  repeat: apply P1-P4;
// abstract from hierarchies of Aspects and Events:
  repeat: apply H4-H5;
// abstract from Aspects:
  repeat: apply A1-A2;
// abstract from hierarchies of Objects:
  repeat: apply H1-H3.
```

Listing 2. Iterative version of the abstraction algorithm

```
// abstract from Aspects:
  repeat: apply P1 to Aspects;
          apply H4 to Aspects;
          apply A1-A2;
// abstract from Events:
  repeat: apply P1 to Events;
          apply H4-H5 to Events;
// abstract from Objects:
  repeat: apply P1-P4 to Objects;
          apply H1-H3.
```

The first version of the algorithm, shown in Listing 1, suggests to abstract from all *Types* almost simultaneously. Abstraction cannot be done in a completely simultaneous manner, since we are not allowed to abstract from taxonomies of *Object Types* before we abstract from *Aspect Types*. The second version, shown in Listing 2, abstracts instead in sequence, first from *Aspect Types* and then from *Event Types*, postponing as much as possible abstraction over *Object Types*.

Determining which version of the algorithm gives better results by producing more meaningful models requires further investigations, which could be aimed at answering the following questions: *(1)* Is there any difference in preferring one algorithm over the other between novel and experienced users? *(2)* Is there any difference in preferring one algorithm over the other depending on the domain and, consequently, on the characteristics of the CM? These empirical questions will be addressed in future work.

5 Final Considerations

In this paper, we have present an extended version of a tested ontology-based model abstraction for ontology-driven conceptual models. Contrary to the original algorithm, our proposal is able to deal more generally with types of Objects, Aspects, Events, and their Parts. We have suggested eight graph-rewriting rules that in compliance with the previously designed Rules `R2`–`R4` can automatically produce an abstracted version of a complex conceptual model. These rules were designed so as to guarantee that they preserve the essential information of the original model by leveraging on the ontological semantics of OntoUML, while simplifying that model.

The research presented here is under active development[7]. As a further goal of the project, we intend to develop a tool that would allow users to interact with their models at different levels of abstraction, thus, making these models more comprehensible. The alternative versions of the algorithm proposed here assume that there is also a possibility of making the abstraction process more user-centred by taking into account the user's current goals and intentions.

Another hypothesis that should be checked is whether the proposed abstractions could help finding errors within the models, i.e., when abstracting leads to unexpected results, this could be a sign of the presence of *anti-patterns* hidden in the original model.

A repository with OntoUML models is under development[8]. We foresee to test different version of our model abstraction algorithm over the models in this repository.

[7] The interested reader can refer to the current version of the Visual Paradigm plugin with supported abstraction functionality on https://github.com/mozzherina/ontouml-vp-plugin.git, and the corresponding server on https://github.com/mozzherina/ontouml-server.git.

[8] https://github.com/unibz-core/ontouml-models.

References

1. Akoka, J., Comyn-Wattiau, I.: Entity-relationship and object-oriented model automatic clustering. Data Knowl. Eng. **20**(2), 87–117 (1996). https://doi.org/10.1016/S0169-023X(96)00007-9
2. Benevides, A., Bourguet, J.R., Guizzardi, G., Peñaloza, R., Almeida, J.: Representing a reference foundational ontology of events in SROIQ. Appl. Ontol. **14**, 1–42 (2019). https://doi.org/10.3233/AO-190214
3. Cook, S., et al.: Unified modeling language (UML) version 2.5.1. Standard, Object Management Group (OMG) (2017). https://www.omg.org/spec/UML/2.5.1
4. Cotnoir, A.J., Varzi, A.C.: Mereology. Oxford Scholarship Online, 1 edn. Oxford University Press, Oxford (2021). https://doi.org/10.1093/oso/9780198749004.001.0001
5. Egyed, A.: Automated abstraction of class diagrams. ACM Trans. Softw. Eng. Methodol. **11**(4), 449–491 (2002). https://doi.org/10.1145/606612.606616
6. Figueiredo, G., Duchardt, A., Hedblom, M.M., Guizzardi, G.: Breaking into pieces: an ontological approach to conceptual model complexity management. In: Proceedings of the 12th International Conference on Research Challenges in Information Science (RCIS), pp. 1–10 (2018). https://doi.org/10.1109/RCIS.2018.8406642
7. Fonseca, C.M., Porello, D., Guizzardi, G., Almeida, J.P.A., Guarino, N.: Relations in ontology-driven conceptual modeling. In: Laender, A.H.F., Pernici, B., Lim, E.-P., de Oliveira, J.P.M. (eds.) ER 2019. LNCS, vol. 11788, pp. 28–42. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-33223-5_4
8. Guarino, N., Guizzardi, G.: "We need to discuss the *Relationship*": revisiting relationships as modeling constructs. In: Zdravkovic, J., Kirikova, M., Johannesson, P. (eds.) CAiSE 2015. LNCS, vol. 9097, pp. 279–294. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19069-3_18
9. Guizzardi, G.: Ontological foundations for structural conceptual models. CITIT PhD.-thesis series 05–74 Telematica Instituut fundamental research series 015, Centre for Telematics and Information Technology, Enschede (2005)
10. Guizzardi, G.: The problem of transitivity of part-whole relations in conceptual modeling revisited. In: van Eck, P., Gordijn, J., Wieringa, R. (eds.) CAiSE 2009. LNCS, vol. 5565, pp. 94–109. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02144-2_12
11. Guizzardi, G., Benevides, A.B., Fonseca, C.M., Porello, D., Almeida, J.P.A., Sales, T.P.: UFO: unified foundational ontology. Appl. Ontol. **17**(1), 1–44 (2021). https://doi.org/10.3233/AO-210256
12. Guizzardi, G., Figueiredo, G., Hedblom, M.M., Poels, G.: Ontology-based model abstraction. In: Proceedings of the 13th International Conference on Research Challenges in Information Science (RCIS), pp. 1–13. IEEE (2019). https://doi.org/10.1109/RCIS.2019.8876971
13. Guizzardi, G., Fonseca, C.M., Almeida, J.P.A., Sales, T.P., Benevides, A.B., Porello, D.: Types and taxonomic structures in conceptual modeling: a novel ontological theory and engineering support. Data Knowl. Eng. **134**, 101891 (2021). https://doi.org/10.1016/j.datak.2021.101891
14. Guizzardi, G., Sales, T.P., Almeida, J.P.A., Poels, G.: Automated conceptual model clustering: a relator-centric approach. In: Software and Systems Modeling, pp. 1–25 (2021)
15. Huang, W., Luo, J., Bednarz, T., Duh, H.: Making graph visualization a user-centered process. J. Visual Lang. Comput. **48**, 1–8 (2018). https://doi.org/10.1016/j.jvlc.2018.07.001

16. Kondylakis, H., Kotzinos, D., Manolescu, I.: RDF graph summarization: principles, techniques and applications. In: Proceedings of the 22nd International Conference on Extending Database Technology (EDBT), pp. 433–436 (2019). https://doi.org/10.5441/002/edbt.2019.38
17. Lombard, L.B.: Events: A Metaphysical Study. Routledge, Abingdon (2019)
18. Lozano, J., Carbonera, J., Abel, M., Pimenta, M.: Ontology view extraction: an approach based on ontological meta-properties. In: IEEE 26th International Conference on Tools with Artificial Intelligence, pp. 122–129 (2014). https://doi.org/10.1109/ICTAI.2014.28
19. Mylopoulos, J.: Conceptual modeling and Telos. In: Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development. Wiley (1992)
20. Pouriyeh, S., et al.: Ontology summarization: graph-based methods and beyond. Int. J. Semant. Comput. **13**(2), 259–283 (2019). https://doi.org/10.1142/S1793351X19300012
21. Verdonck, M., Gailly, F.: Insights on the use and application of ontology and conceptual modeling languages in ontology-driven conceptual modeling. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 83–97. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_7
22. Verdonck, M., Gailly, F., Pergl, R., Guizzardi, G., Souza, B.F.M., Pastor, O.: Comparing traditional conceptual modeling with ontology-driven conceptual modeling: an empirical study. Inf. Syst. **81**, 92–103 (2019). https://doi.org/10.1016/j.is.2018.11.009
23. Villegas Niño, A.: A filtering engine for large conceptual schemas. Ph.D. thesis, Universitat Politècnica de Catalunya (2013)