Formal Ontology in Information Systems
N. Aussenac-Gilles et al. (Eds.)

doi:10.3233/FAIA231122

Inferring Ontological Categories of OWL Classes Using Foundational Rules

Pedro Paulo F. BARCELOS [a,1], Tiago Prince SALES [b], Elena ROMANENKO [a], João Paulo A. ALMEIDA [c], Gal ENGELBERG [d], Dan KLEIN [d] and Giancarlo GUIZZARDI [b]

[a] *KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano, Bolzano, Italy*
[b] *Semantics, Cybersecurity & Services (SCS), University of Twente, Enschede, The Netherlands*
[c] *Ontology & Conceptual Modeling Research Group (NEMO), Federal University of Espírito Santo, Vitória, Brazil*
[d] *Accenture Labs, Israel*

Abstract. Several efforts that leverage the tools of formal ontology (such as OntoClean, OntoUML, and UFO) have demonstrated the fruitfulness of considering key metaproperties of classes in ontology engineering. These metaproperties include sortality, rigidity, and external dependence, and give rise to many fine-grained ontological categories for classes, including, among others, kinds, phases, roles, mixins, etc. Despite that, it is still common practice to apply representation schemes and approaches—such as OWL—that do not benefit from identifying these ontological categories, and simplistically treat all classes in the same manner. In this paper, we propose an approach to support the automated classification of classes into the ontological categories underlying the (g)UFO foundational ontology. We propose a set of inference rules derived from (g)UFO's axiomatization that, given an initial classification of the classes in an OWL ontology, can support the inference of the classification for the remaining classes in the ontology. We formalize these rules, implement them in a computational tool and assess them against a catalog of ontologies designed by a variety of users for a number of domains.

Keywords. Applications and Methods, OWL, UFO, gUFO, Ontology Grounding, Foundational Ontologies, Ontology Tools

Introduction

One of the key motivations behind the original idea of the Semantic Web was to provide assistance for a sort of Semantic Interoperability. The idea was to offer support for complex information services by combining information sources that have been designed in a concurrent and distributed manner. In this context, ontologies—which had been since the '90s used for this purpose in the Knowledge Exchange Community [1]—were immediately identified as one of the enabling technologies.

[1]Corresponding Author: Pedro Paulo F. Barcelos, Piazza Università, 1, 39100, Bolzano, Italy; E-mail: pfavatobarcelos@unibz.it.

Over the years, ontologies have been promoted both as a **(purpose 1)** reference model of consensus to support reuse and interoperability of domain conceptualizations or as an **(purpose 2)** explicit, declarative, and machine-processable artifact coding a domain model to enable automated reasoning. This duality, however, points to different (and even conflicting) sets of quality criteria and requirements that these artifacts and the representation languages in which they are expressed should meet.

To serve **purpose 1**, ontologies should be constructed in ways that maximize the expressivity in capturing fundamental aspects of the underlying domain and in making explicit the underlying ontological commitments. In contrast, to support **purpose 2**, they should be built in a way that supports decidable and computationally tractable automated reasoning. As it is well known, there is a trade-off between expressivity and maintaining these desirable computational properties [2]. In the context of the Semantic Web, the Web Ontology Language (OWL) was designed to maximize the latter side of this trade-off and, despite its limited expressivity (when compared to the First Order [3] or Higher Order Logics [4] that have been used in ontology representation), OWL has been very successful in giving rise to a plethora of specifications (often called lightweight ontologies) in a large variety of domains.

As shown by several authors, these OWL ontologies do not always live up to the promise of promoting either interoperability or reuse in complex and critical domains [5–9]. The reason for that goes beyond mere logical expressivity. As beautifully expressed by Varzi's dictum: *"No ontology without Ontology"* [10], to support **purpose 1** ontologies should be constructed with a fuller consideration of (the discipline of) Ontology, i.e., with the explicit support of true ontological theories.

To offer that kind of foundational support for the construction of OWL ontologies, some of us have proposed gUFO [11]. Standing for *gentle UFO*, gUFO is an OWL superstructure implementing the Unified Foundational Ontology (UFO) [12]. By partially incorporating UFO's ontological distinctions and regulating logical constraints (the ones that can be expressed in OWL), gUFO can support the systematic alignment of classes in OWL ontologies to its foundational categories, increasing the quality of these ontologies (in the sense of **purpose 1**) by design.

However, there is a vast amount of OWL ontologies that have been built without such support, and expecting the ontologies to be manually re-engineered is unrealistic. This is the exact problem we address in this paper. We propose a set of inference rules derived from UFO's and gUFO's axiomatization that, given an initial seeding, i.e., an initial classification of the classes in an OWL ontology into gUFO's foundational categories, are triggered to infer the gUFO classification for the remainder of the classes. Further, we implement these rules in a computational tool that operates in an automatic or semi-automatic manner, i.e., automatic whenever possible and by interacting with the ontologist when not. Finally, we test this tool against a catalog of (g)UFO-based ontologies that collects models designed by a variety of users and domains.

The remainder of the paper is structured as follows: section 1 briefly presents the background of the paper, namely, UFO and gUFO; section 2 introduces our inference rules; section 3 discusses the proposed computational tool; section 4 presents and discusses our empirical work, validating the rules and tool; section 5 discusses related work; section 6 presents some final considerations.

1. Background: UFO and gUFO

1.1. The Unified Foundational Ontology (UFO)

The Unified Foundational Ontology (UFO) [12] is an axiomatic domain-independent formal theory that builds on contributions from analytic metaphysics, cognitive sciences, linguistics, and philosophical logic. It results from the integration and re-visitation of previous foundational approaches, such as OntoClean [13], DOLCE [14], and GFO [15]. UFO addresses fundamental ontological notions via a set of micro-theories addressing types and taxonomic structures, part-whole relations, relations, events, dependence, and multilevel modeling, among others. Over the years, it has been successfully used to model a wide range of domains [12], including risk, services, legal relations, telecommunications, trust, petroleum and gas, genomics, and many more. According to a recent study [16], it is the second most used foundational ontology in conceptual modeling and the one with the fastest adoption rate.

As a 3D ontology, UFO accounts for the existence of both endurants and perdurants. Endurants are object-like individuals. They endure in time and are able to qualitatively change while maintaining their identities [3]. Examples include ordinary objects of everyday life, such as a person, a house, and a car; particularized relational properties, such as a marriage, a rental contract, and one's love for another; and existentially dependent aspects of objects, such as the weight of a car, the language skills of a person, and the color of a house. Endurants are sometimes called continuants or simply objects. In contrast, perdurants are entities that unfold in time, accumulating temporal parts. Examples include actions, such as walking and taking a photo, but also natural occurrences, such as rain and the fall of a meteor.

Endurant types in UFO can be of different sorts, distinguished by formal metaproperties such as *rigidity* and *sortality*. Rigidity is a property of a type that describes the dynamics of how it may be instantiated. From this perspective, types may be classified as rigid, anti-rigid, and semi-rigid. Rigid types are those that classify their instances necessarily (in the modal sense). Stating that the type "Person" is rigid means that no person may cease to be a person and still exist. Anti-rigid types are those that classify their instances contingently, i.e., their instances can move in and out of their extension without altering their identity. Examples include the types "Child", "Student", and "Customer". Lastly, semi-rigid types are those that classify some of their instances necessarily and some of their instances contingently. For example, the type "Female Animal" is semi-rigid because it applies necessarily to some animals, e.g., lionesses and cows, and contingently to others, such as clownfishes and starfishes of certain species (which may change sex given certain conditions). Given the formalization of these notions in UFO, it follows that: *(i)* every type is either rigid, anti-rigid, or semi-rigid (cf. theorem t5 of [3]); *(ii)* no type is both rigid and anti-rigid, rigid and semi-rigid, and anti-rigid and semi-rigid (cf. theorem t6 of [3]); *(iii)* there is no rigid type that specializes an anti-rigid type (cf. theorem t7 of [3]); and *(iv)* there is no semi-rigid type that specializes an anti-rigid type (cf. theorem t8 of [3]).

To explain sortality, we must first introduce the notion of a principle of identity. A principle of identity makes explicit the properties that no two individuals can have in common, since such properties uniquely identify them. In particular, it also informs which changes an individual can undergo without changing its identity. The relation be-

tween a type and the principles of identity of its instances is defined by its sortality. A type is a sortal if all of its instances follow the same identity principle. Examples include the types "Person", "Woman", and "Child". If a sortal type *supplies* an identity principle to its instances, it is deemed an ultimate sortal—or simply a kind. Everyday examples of kinds are the types "Person" and "Organization". As a complement to sortals, we have non-sortal types. A non-sortal type aggregates properties that are common to different sortals and, thus, can be instantiated by individuals that follow different identity principles. An example of such a type is "Work of Art", which applies to paintings, music compositions, and statues.

In terms of sortality, the UFO adopts the following axioms. Every individual must follow a unique identity principle and thus instantiate exactly one kind (cf. axioms a9 and a10 of [3]). For every sortal type T, there is one kind K such that all instances of T instantiate K (cf. axiom a11 of [3]). No type is both a sortal and a non-sortal (cf. axiom 12 of [3]). Some useful theorems in this respect for our purpose in this paper are: *(i)* every sortal specializes a unique kind (cf. theorem t13 and t14 of [3]); *(ii)* a non-sortal cannot specialize a sortal (cf. theorem t15 of [3]); and *(iii)* non-sortals do not have direct instances, their instances are also instances of a sortal that either specializes the non-sortal, or that specializes a common non-sortal supertype (cf. theorem t16 of [3]).

By combining the sortality and rigidity aspects, we obtain the leaf categories of endurant types, namely:

- **kind**: a rigid sortal that supplies an identity principle to its instances. For example, "Dog", "Car", "Person", and "Organization";
- **subkind**: a rigid sortal that is not a kind. For example, "Sedan" is a subkind of "Car", "Bulldog" is a subkind of "Dog";
- **phase**: an anti-rigid sortal whose dynamic instantiation condition depends on an intrinsic property, e.g., a "Dog" instantiates the phase "Puppy" if it is at most 2 years old;
- **role**: an anti-rigid sortal whose dynamic instantiation condition depends on a relational property, e.g., a "Person" instantiates the role "Student" when enrolled in at least one educational institution;
- **category**: a rigid non-sortal type. For example, "Furniture", "Agent", "Vehicle";
- **phase mixin**: an anti-rigid non-sortal whose dynamic instantiation condition depends on an intrinsic property, e.g., "Infant Animal";
- **role mixin**: an anti-rigid non-sortal whose dynamic instantiation condition depends on a relational property, e.g., "Customer", "Insured Item", "Service Provider";
- **mixin**: a semi-rigid non-sortal type. For example, "Music Artist", a type that rigidly characterizes bands and contingently characterizes people.

1.2. gUFO (gentle UFO): A Lightweight Implementation of UFO in OWL

gUFO is a lightweight implementation of UFO designed for the Semantic Web [11]. The term *lightweight* is used for three reasons. First, because of the limited expressivity of OWL, gUFO does not contain the complete axiomatization of UFO. For instance, we cannot formally define a class as being anti-rigid, as there is no alethic modality in OWL. Second, gUFO does not completely implement all the micro-theories defined in UFO. In

particular, the theory of perdurants is only minimally implemented to support the representation of part-whole relations between events, the participation of objects in events, and the historical dependence between events. Third, given its application-oriented focus, some pragmatic implementations were made, such as terminology simplification (e.g., perdurants are called events instead).

gUFO was designed to offer a conceptual base on which ontologists can build their ontologies, offering several patterns to cope with recurrent conceptual issues (e.g., the modeling of roles with multiple disjoint allowed types). In addition, it also provides patterns that support the working ontologist to cope with recurrent implementation challenges, such as the representation of temporal data (e.g., the fact that Jeff Bezos used to be the CEO of Amazon, but is not anymore) and the representation of property values in multiple scales (e.g., a dog's weight in pounds and kilograms).

To reuse gUFO, one may instantiate and/or specialize the various classes, object properties, and data properties it provides. A key feature of gUFO is that it includes two class taxonomies: one with classes whose instances are individuals, such as `gufo:Endurant` and `gufo:Object`; and another with classes whose instances are types, such as `gufo:Kind`, `gufo:Phase`, and `gufo:Category`. Considering these two taxonomies, gUFO may be reused in the four ways shown in Listing 1 (using Turtle as notation).

```
# 1. By instantiating classes in the taxonomy of individuals
:John rdf:type gufo:Object .
:JohnAndMaryMarriage rdf:type gufo:Relator.

# 2. By specializing classes in the taxonomy of individuals
:Person rdfs:subClassOf gufo:Object .
:Marriage rdfs:subClassOf gufo:Relator .

# 3. By instantiating classes in the taxonomy of types
:Person rdf:type gufo:Kind .
:Child rdf:type gufo:Phase .

# 4. By specializing classes in the taxonomy of types
:PersonPhase rdfs:subClassOf gufo:Phase .
:AcademicRank rdfs:subClassOf gufo:Role .

# Combining strategies 2 and 3
:Person rdf:type owl:Class, gufo:Kind ;
        rdfs:subClassOf gufo:Object .
```

Listing 1: The four ways in which ontologies can reuse the classes defined in gUFO.

Note that these four representation strategies are not mutually exclusive, but complementary. For example, as shown in the last lines of Listing 1, ontologies can use strategies 2 and 3 in combination. This particular representation strategy employs OWL 2 punning, allowing the interpretation of the same identifier as two different model entities, such as classes, object properties, or individuals. In our example, we state that the class `:Person` instantiates the class `gufo:Kind`.

2. Inference Rules to Categorize Endurant Types

Our objective in this paper is to develop a system to infer the UFO meta-categories of classes in an OWL ontology given an initial seeding. More precisely, consider ET as the set of leaf classes defined in gUFO's taxonomy of endurant types, namely `gufo:Kind`, `gufo:Subkind`, `gufo:Role`, `gufo:Phase`, `gufo:Category`, `gufo:RoleMixin`, `gufo:PhaseMixin`, and `gufo:Mixin`. Given an OWL ontology O containing a set of classes C and an initial mapping M (*seeding*) between classes in C and classes in ET using `rdf:type`, what additional `rdf:type` mappings between classes of C and ET can we infer?

Naturally, this task would be significantly facilitated if the complete axiomatization of UFO was implemented in gUFO. Since this is not the case, we propose a set of rules based on UFO's [3] and gUFO's axiomatizations [11]. In the following, we specify our inference rules in First Order Logic (FOL). Consider that all free variables are universally quantified, having all their occurring formulas as their scope. Moreover, we assume that our domain of quantification only includes types.

We start by declaring our *subClassOf* relation, which is reflexive (R01) and transitive (R02), directly corresponding to the `rdfs:subClassOf` property and compatible with the UFO's *specialization* relation.

R01: $subClassOf(x,x)$
R02: $subClassOf(x,y) \wedge subClassOf(y,z) \rightarrow subClassOf(x,z)$

Rules R03–R16 were mapped from gUFO's taxonomy of endurant types. Capitalized unary predicates (e.g., *Sortal*, *Kind*, *Category*) directly correspond to gUFO's classes in its taxonomy of types, implications correspond to specialization relations, and equivalence and exclusive disjunctions are used to map disjoint unions.

R03: $EndurantType(x) \leftrightarrow RigidType(x) \oplus NonRigidType(x)$[2]
R04: $NonRigidType(x) \leftrightarrow AntiRigidType(x) \oplus SemiRigidType(x)$
R05: $EndurantType(x) \leftrightarrow Sortal(x) \oplus NonSortal(x)$
R06: $Kind(x) \rightarrow RigidType(x) \wedge Sortal(x)$
R07: $SubKind(x) \rightarrow RigidType(x) \wedge Sortal(x)$
R08: $\nexists x(Kind(x) \wedge SubKind(x))$
R09: $Role(x) \rightarrow AntiRigidType(x) \wedge Sortal(x)$
R10: $Phase(x) \rightarrow AntiRigidType(x) \wedge Sortal(x)$
R11: $\nexists x(Phase(x) \wedge Role(x))$
R12: $Category(x) \rightarrow NonSortal(x) \wedge RigidType(x)$
R13: $RoleMixin(x) \rightarrow NonSortal(x) \wedge AntiRigidType(x)$
R14: $PhaseMixin(x) \rightarrow NonSortal(x) \wedge AntiRigidType(x)$
R15: $\nexists x(PhaseMixin(x) \wedge RoleMixin(x))$
R16: $Mixin(x) \rightarrow NonSortal(x) \wedge SemiRigidType(x)$

For our purpose, we need to complement these rules with R17–R21, which require endurant types to instantiate at least one of the leaf types in gUFO's taxonomy of types. This is consistent with what is achieved by axioms a13–a21 in the original theory [3], except for R19, which is more restrictive than in the source formalization. In R19, we equate semi-rigid types with mixins, which are non-sortal semi-rigid types. We do that because gUFO does not explicitly account for the existence of semi-rigid sortals.

R17: $RigidType(x) \rightarrow Category(x) \vee Kind(x) \vee SubKind(x)$

[2]The symbol $\oplus$ is used here to represent an exclusive or operator and $\nexists$ as a shorthand for $\neg\exists$.

R18: $AntiRigidType(x) \rightarrow Role(x) \vee Phase(x) \vee RoleMixin(x) \vee PhaseMixin(x)$
R19: $SemiRigidType(x) \rightarrow Mixin(x)$
R20: $Sortal(x) \rightarrow Kind(x) \vee Phase(x) \vee Role(x) \vee SubKind(x)$
R21: $NonSortal(x) \rightarrow Category(x) \vee PhaseMixin(x) \vee RoleMixin(x) \vee Mixin(x)$

As gUFO does not implement any restrictions regarding the rigidity of an endurant type and of the types it specializes and generalizes, we implement rules R22–R25 based on UFO's original formalization of [3]. R22 is based on UFO's theorem t7, which states that no rigid type can specialize an anti-rigid type. R23 is based on theorem t8, a similar theorem for semi-rigid types. R24 and R25 are not explicitly formalized in UFO but are implied by it. R24 states that for every anti-rigid sortal type x that specializes a category y, a rigid non-sortal type, there is at least a rigid sortal type z of which x is a subclass and which is a subclass of y. R25 states that for every mixin, there is at least one rigid type and one anti-rigid type that specialize it.

R22: $RigidType(x) \wedge subClassOf(x,y) \rightarrow \neg AntiRigidType(y)$
R23: $SemiRigidType(x) \wedge subClassOf(x,y) \rightarrow \neg AntiRigidType(y)$
R24: $subClassOf(x,y) \wedge AntiRigidType(x) \wedge Sortal(x) \wedge Category(y) \rightarrow \exists z(subClassOf(x,z) \wedge subClassOf(z,y) \wedge RigidType(z) \wedge Sortal(z))$
R25: $Mixin(x) \rightarrow \exists y,z(subClassOf(y,x) \wedge RigidType(y) \wedge subClassOf(z,x) \wedge AntiRigidType(z))$

gUFO also does not formalize any constraints regarding sortality and taxonomic structures. Based on UFO's formalization, we implement those on rules R26–R28. R26 states that every type that a kind specializes (which is not itself) is a non-sortal. R27 states that non-sortal types can only specialize non-sortal types (based on UFO's theorem t15 [3]). R28 states that every sortal must specialize a unique kind (based on UFO's theorems t13 and t14).

R26: $x \neq y \wedge Kind(x) \wedge subClassOf(x,y) \rightarrow NonSortal(y)$
R27: $NonSortal(x) \wedge subClassOf(x,y) \rightarrow NonSortal(y)$
R28: $Sortal(x) \rightarrow \exists! y(subClassOf(x,y) \wedge Kind(y))$

To implement UFO's theorem t16 [3], which states that non-sortals do not have direct instances and classify individuals of at least two different kinds, we need a rule stating that every non-sortal must be a superclass (or a sibling) of at least two sortals that specialize different kinds. To do that, we first define the relations *shareKind* and *shareSuperClass*. R29 states that the types x and y share a kind if and only if there is a single kind z such that both x and y specialize it. R30 states that the type x shares a same superclass with the type y if and only if there is at least one type that both x and y specialize. Then, using R29 and R30, R31 implements our version of theorem t16.

R29: $shareKind(x,y) \leftrightarrow \exists! z(Kind(z) \wedge subClassOf(x,z) \wedge subClassOf(y,z))$
R30: $shareSuperClass(x,y) \leftrightarrow \exists z(subClassOf(x,z) \wedge subClassOf(y,z))$
R31: $NonSortal(x) \rightarrow \exists y,z(y \neq z \wedge Sortal(y) \wedge Sortal(z) \wedge \neg shareKind(y,z) \wedge (subClassOf(y,x) \vee shareSuperClass(x,y)) \wedge (subClassOf(z,x) \vee shareSuperClass(x,z)))$

Rules R32–R34 stem from the constraint that specializations of roles and role mixins always inhere their parent's *relational dependency*. So, phases cannot specialize roles and role mixins (R32) (otherwise they would be relationally dependent), and, likewise, that phase mixins cannot specialize role mixins (R33). Further, whenever a role specializes a phase mixin, it does that by specializing a phase that specializes that phase mixin (R34).

R32: $Phase(x) \wedge subClassOf(x,y) \rightarrow \neg Role(y) \wedge \neg RoleMixin(y)$
R33: $PhaseMixin(x) \wedge subClassOf(x,y) \rightarrow \neg RoleMixin(y)$
R34: $Role(x) \wedge PhaseMixin(y) \wedge subClassOf(x,y) \rightarrow$
$\exists z(Phase(z) \wedge subClassOf(x,z) \wedge subClassOf(z,y))$

Finally, rule R35 specifies that phases must always have at least one sibling class sharing a common kind, while rules R36 and R37 state that phase mixins must always have at least one sibling sharing a common category. Note, that these are weaker constraints than those of the original theory, which require phases and phase mixins to occur in partitions of kinds and categories respectively. (A set of types S is a partition of type T if all types in S are pairwise disjoint and specialize T, and every instance of T is an instance of a type in S.)

R35: $Phase(x) \rightarrow \exists y(Phase(y) \wedge shareKind(x,y) \wedge \neg subClassOf(x,y) \wedge \neg subClassOf(y,x))$
R36: $PhaseMixin(x) \rightarrow \exists y(Category(y) \wedge subClassOf(x,y))$
R37: $PhaseMixin(x) \wedge Category(y) \wedge subClassOf(x,y) \rightarrow \exists z(PhaseMixin(z) \wedge \neg subClassOf(x,z) \wedge$
$\neg subClassOf(z,x) \wedge subClassOf(z,y))$

To verify the consistency and validate if the rules were sufficient for our inference needs (i.e., the completeness of the rule set), we implemented the specified rules in Alloy [17], a logic language that supports bounded verification. The language is accompanied by a solver tool, called Alloy Analyzer, which allowed us to, in a fixed scope, generate instances that comply with our rule set and to seek counter-examples that would invalidate expected theorems from it. The implementation of our inference rules in Alloy is available in a git repository (https://purl.org/scior/alloy).

3. Tool Support

To assess the effectiveness of our inference rules in practice and make them available to working ontologists, we implemented them in an open-source command-line Python tool called **Scior** (see https://purl.org/scior for the source code repository). Two main requirements guided the design of Scior. First, users should pick whether Scior should reason with a closed-world assumption (CWA) or an open-world assumption (OWA) [18]. When it reasons with a close-world assumption, it presumes that all classes and properties of an ontology have been declared. So, if a class has no subclass, this means that there is not one. Conversely, when it reasons with an open-world assumption, it presupposes that not all classes and properties of an ontology have been declared. So, for any class in the ontology, there might be a class that specializes it (or which it specializes) that just has not been declared.

Second, Scior should support the alignment of an ontology to gUFO in a semi-automated way. This means that, whenever the rules and the declared alignments are sufficient to infer a new alignment, Scior should make it. Otherwise, Scior should provide users with options to manually declare an alignment themselves, but discard possibilities that were rejected by the inference rules. For instance, we only know that a class C specializes a class K that instantiates `gufo:Kind`, Scior should conclude that C is either an instance of `gufo:Role`, `gufo:Phase`, or `gufo:SubKind`, and only offer the user these options. It must be highlighted that when Scior is in an OWA context and operating in its automatic mode, if it identifies a single declared class that can fulfill one of its rules, it shall assign the rule's required classification to that class, even though the OWA may consider that other non-declared classes could be possible.

4. Evaluation

We evaluate Scior from two perspectives, correctness and effectiveness, which are described in subsections 4.2 and 4.3 respectively.

4.1. Materials

To conduct our evaluation, we needed a set of taxonomies for which the ontological categorization of its classes was known in advance. Since no such dataset existed, we built one from the models available in the FAIR Model Catalog for Ontology-Driven Conceptual Modeling Research, short-named OntoUML/UFO Catalog [19]. This open-source catalog contains ontology models grounded in UFO and in its derived conceptual modeling language, OntoUML. Its purpose is to support empirical research by providing high-quality, curated, structured, and machine-processable data on UFO-based models. It offers a diverse collection of models, created by modelers with varying modeling skills, for a range of domains, and for different purposes. The catalog provides a serialization of these models in Turtle, allowing them to be handled by Semantic Web manipulation libraries and frameworks.

To assemble our test dataset of taxonomies, we collected 140 models that compose the catalog. We mapped each OntoUML model into an OWL ontology, keeping only their classes and generalizations. Each class in the source model was mapped into an `owl:Class` and each generalization was mapped into an `rdfs:subClassOf` statement. In a separate file, we also mapped OntoUML's stereotypes into gUFO types, focusing on endurant types. This mapping is described as follows.

- Classes with the following OntoUML stereotypes receive the gUFO homonym classifications: `gufo:Category`, `gufo:Mixin`, `gufo:Phase`, `gufo:PhaseMixin`, `gufo:Kind`, `gufo:SubKind`, `gufo:Role`, and `gufo:RoleMixin`.
- Classes with OntoUML stereotypes that represent ultimate sortals, namely Collective, Quality, Quantity, Mode, and Relator, are mapped to `gufo:Kind`.
- The OntoUML stereotype HistoricalRole is mapped to `gufo:Role`, and HistoricalRoleMixin is mapped to `gufo:RoleMixin`.
- Finally, classes without OntoUML stereotypes or with stereotypes not cited here receive the string "other", labeling that they are out of scope for future exclusion from our test dataset.

In these OWL ontologies, we performed a procedure named taxonomical graphs separation. In this procedure, the composing taxonomies of each ontology were separated into different graphs, here called *taxonomical graphs*. The algorithm for this procedure is as follows.

1. A list of root nodes is generated.
2. For the first element of that list, all other nodes that are reachable from it (i.e., nodes that can be accessed via the specialization/generalization relations) are separated into a new taxonomical graph, and their information is isolated in a new file. The root node selected as a pivot is removed from the root list. All its reachable nodes are also removed from there.
3. Step 3 is repeated until the root list is empty.

Applying the taxonomical graph separation procedure to the previously selected models from the OntoUML/UFO catalog and excluding the results that contained out-of-scope classes resulted in 656 taxonomies containing a sum of 5108 classes.

4.2. Correctness Evaluation

We qualitatively evaluated the correctness of the rules implemented in Scior in three steps. First, during development, we performed a series of manual tests using toy examples. For each implemented rule, we created several positive and negative tests to verify if Scior inferred the expected category when it was supposed to and that it did not infer when it was not supposed to.

The second step consisted of running Scior on all of our taxonomies and assessing if the inferred categories matched the ones provided in the dataset. For this, we executed Scior once per class in each taxonomy in the dataset, providing this class as the single seed. Then, we randomly selected a sample of these executions and manually assessed if the inferred knowledge was consistent with the original data, even if a definite categorization was not possible. Additionally, we verified a sample of executions for which we knew the inferred classes did not match the original ones. This was done independently by two authors of this paper. For both, all the outputs matched the expected behavior.

The last and third evaluation step comprised executing Scior again over all 656 taxonomies and checking to which ones it reported inconsistencies. The list of inconsistent taxonomies must match the one detected when using the queries presented by Guizzardi et al. [3]. They provided seven different SPARQL queries to detect violations of the rules imposed by the UFO's types formal theory, more specifically, regarding entities' sortality and rigidity. As their rules do not cover relational dependency (rules R32–R34) and the necessity of partition sets for phases and phase mixins (rules R35 and R37), we created additional SPARQL queries for these specific verifications. For a CWA context, all rules must not report violations, while for OWA the models do not need to comply with the `sortalMustSpecializeUltimateSortal` constraint and with the queries that verify this work's rules R35 and R37.

Performing the evaluation, we found that a single taxonomy (0.15%) did not match the expected behavior for OWA models, while 23 (3.51%) did not match for CWA models. Analyzing the results, we concluded that the OWA divergence happened because the queries presented by Guizzardi et al. could not capture the invalid situation (according to R26) of a kind specializing a subkind that does not have another kind as its superclass. For the CWA models, all unexpected results were caused by the query that verifies the `nonSortalMustHaveSortalSpecialization` constraint, which is less restrictive than this work's rule R31. Here, we require non-sortals to be specialized by at least two sortals carrying different principles of identity. We could then conclude that Scior presented the expected results for all 656 tested taxonomies.

4.3. Effectiveness Evaluation

We quantitatively evaluate the effectiveness of our inference rules by measuring the degree to which Scior can infer, given an initial seeding, the categories of the remaining classes in a taxonomy specified in OWL.[3] We do that by running Scior on different tax-

[3]The tested dataset and resulting data are available at https://purl.org/scior/fois2023.

onomies, with varying levels of seedings, and under both open and closed-world assumption modes.

4.3.1. Methods

More precisely, for each taxonomy, we repeatedly test Scior by providing increasing levels of seeding, starting at 10% of the classes and making increments of 10% until we seeded 90% of the taxonomy. For each percentage level, we run Scior 10 times, each time providing a different randomly selected subset of categorized classes as the seeding. We run these tests first with Scior on open-world assumption and then on closed-world assumption. We executed our tests in this way to minimize the possibility that any taxonomy or seed would affect our results, as well as to observe how the tool behaves when the level of seeding changes.

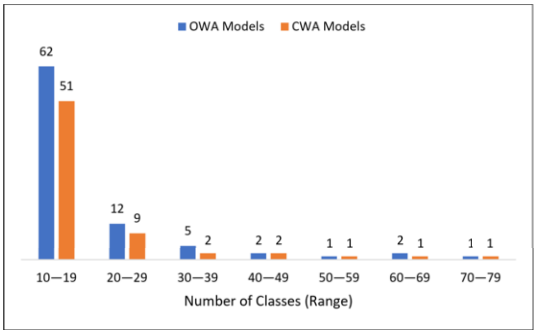
For each execution of Scior, we registered how many endurant types' leaf categories could be excluded from the list of classifications a class may receive; i.e., we measured which classifications from the following list could be automatically discarded: kind, subkind, role, phase, category, role mixin, phase mixin, or mixin. Hence, results vary from zero, indicating that no information could be inferred, to seven, representing that Scior could determine the exact classification of the evaluated class.

As we wanted to execute Scior multiple times over numerous taxonomies, we developed a testing software to help us, dubbed Scior-Tester.[4] We configured this tester to always execute Scior in its automatic mode, thus avoiding any user interaction. As Scior must be applied to unclassified classes, the tester removes every gUFO classification from the taxonomies, preserving them elsewhere for later use.

We executed this procedure over a subset of the 656 taxonomies, excluding those that had fewer than 10 classes with their ontological categories mapped and that violated UFO's axiomatization. These violations were caused, for instance, by syntactical errors in the source OntoUML models or because the models were developed in a previous version of OntoUML. Because of these two filters, we reached a dataset of 85 taxonomies (with 1624 classes). Note, however, that we could only use all 85 taxonomies when evaluating Scior running with an open-world assumption. That is because several OntoUML models we reused were not built assuming a closed-world scenario. Excluding taxonomies originating from such ontologies led us to a second dataset of 67 taxonomies (with 1234 classes) to be used when evaluating Scior running with a closed-world assumption.

The taxonomy sets used for testing OWA and CWA models share many similarities concerning their measures, with the former having approximately 19 classes on average and the former having approximately 18 classes on average. Figure 1 depicts the distribution of these taxonomies in terms of their numbers of classes. The taxonomies selected for open-world testing are represented in blue and termed "OWA Models" and those suitable for closed-world testing are represented in orange and termed "CWA Models". Notice that there are significantly more values on the left-hand side of the chart, that is, with 30 or fewer classes.

[4] https://purl.org/scior/tester



Measures	OWA Models	CWA Models
Population size	85	67
Mean (μ)	19.11	18.42
Median	14	14
Modes	10 11	10
Lowest value	10	10
Highest value	76	76
Interquartile range	8	8
First quartile	12	11
Third quartile	20	19
Variance (σ2)	164.33	158.06
Standard deviation (σ)	12.82	12.57
Quartile deviation	4	4
Mean Absolute Deviation	8.6	7.82

Figure 1. Distribution of the taxonomies used in the evaluation according to their number of classes (left) and their corresponding dispersion measures (right).

4.4. Results and Discussion

The results obtained for OWA and CWA models are summarized in Figure 2 and Figure 3, respectively. For each level of initial seeding (from 10% to 90%, laid out horizontally), we show the portion of classes *(i)* that did not have classifications excluded (in red), *(ii)* that had at least one and at most four classifications excluded (in gray), *(iii)* that had exactly five (in blue) and *(iv)* exactly six categories excluded (in yellow), and, finally, *(v)* the classes for which Scior could determine the applicable leaf category (in green). Results are averages for 10 executions of Scior with a randomly selected subset of categorized classes for seeding.

The more classes excluded, the more effective is our proposed set of rules. Results in which no classification could be excluded indicate that Scior could not infer any ontological property for a class and that no new knowledge was added to it, while results that exclude seven categories determine the final classification of a class. Intermediate results, where at least one and at most six categories were excluded, contribute to the overall increase of knowledge as they reduce the number of possibilities users have when manually deciding the final classification for a class, leading to an easier and less error-prone decision-making process.

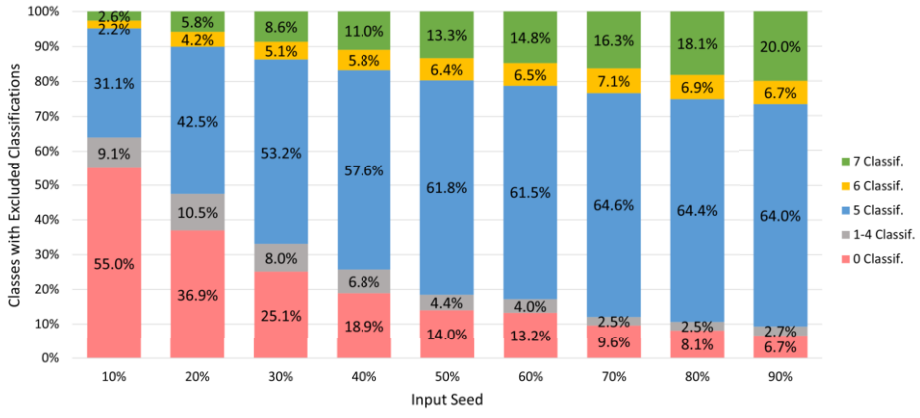


Figure 2. Results of the effectiveness evaluation performed on OWA Models.

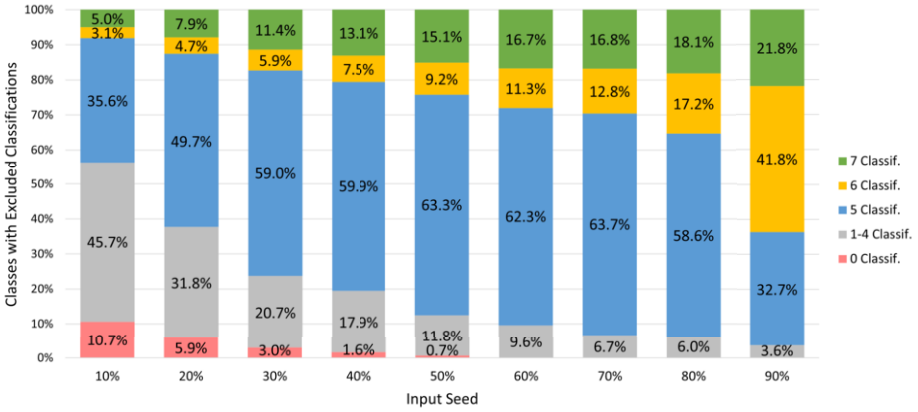


Figure 3. Results of the effectiveness evaluation performed on CWA Models.

These results reveal a clear correlation between the amount of input provided and the higher number of excluded categories for both CWA and OWA models. The charts exhibit (as expected) better classification results for CWA models than for OWA ones. Even though the total exclusion of possibilities is just slightly higher for the former, the higher percentages of exclusion of intermediate numbers of categories are noteworthy. Another interesting result is the expressive portion of cases that lead to the exclusion of five categories. Such a situation occurs when Scior identifies that a class must be a non-kind sortal (i.e., must be a role, phase, or subkind).

By presenting and analyzing the results of the test, we could evidence the effectiveness of Scior and, therefore, of the proposed inference rules for ontological categorization in both open and closed-world assumption situations. When considering closed-world applications, Scior presented stronger results, always excluding some categories when more than 50% of classes are provided as seeds. Although OWA models had on average 20.0% of their classes fully classified for a seeding level of 90%, they still present a percentage of zero exclusions for this amount of input (6.7%). In both assumptions, the percentage of models with at least five categories excluded is higher than 50% for a seeding level of 20%. Note that these results refer to the fully automated application of Scior, without prompting the user for additional classification information beyond the original seeding. In its semi-automated mode, partial classifications also support the user by narrowing the number of alternatives for manual classification.

5. Related Work

We focus here on works that define a mapping between OWL and foundational ontologies. An example is the work by Krasnoperova [20], which uses Bunge's ontology to develop modeling rules and guidelines on how to better represent real-world domains in OWL. Different from our work, the author proposed specific rules on how to model real-world domain elements in OWL ontologies.

In a nonsystematic literature review, we could find examples of alignments to foundational ontologies different from UFO. For instance, Keet et al. [21] proposed a method

leveraging a decision tree for DOLCE categories and an automated reasoner to simplify the challenge that ontology developers face regarding the selection of the ontological category for their models. Another work that tries to bridge this gap between OWL and foundational ontology concepts is the BFO Classifier [22]. This software contains a decision diagram for BFO and offers modelers a series of questions regarding identifying the suitable BFO entity to align with the proper domain entity. In any case, both DOLCE and BFO do not address sortality and rigidity of classes.

Some works rely on OntoClean [13] to formally evaluate taxonomic relations in OWL ontologies. A work with similarities to ours that uses OntoClean is AEON [23], a software that automatically tags domain concepts according to metaproperties, checking to satisfy the predefined constraints for discovering taxonomic errors. Different from Scior—which does not consider domain information for categorizing ontology classes—AEON uses the Web as a source of knowledge, searching for patterns that indicate how to tag concepts properly. Hence, the proposals differ in the sense that the mapping we propose is done through inferences provided by (g)UFO, while AEON uses OntoClean's rules for constraint checking. We should also highlight that UFO's metaproperties regarding endurant types extend the original one proposed by OntoClean, providing a formalization that deals with a richer system of meta-categories [3,24].

Finally, we mention that the rules we defined in this paper were inspired by the conversion of the UFO axioms and theorems to OntoUML, especially by the ones specified in Guizzardi et al. [3], which define constraints for taxonomical structures of endurant types. Besides the different scope—they create a set of rules for a conceptual modeling language and not for a computational ontology—our set of rules is broader, containing rules generated from these axioms and from others that are associated with relational dependency and with the necessity of partition sets. The authors also provided SPARQL queries based on the original formalization of the rules. However, their intention differs from ours, as the queries are designed to detect anti-patterns in the user's ontologies caused by violation of the rules (and are not aimed at inferring ontological categories).

6. Final Remarks

In this paper, we presented a set of inference rules that (semi-)automatically support users in grounding their existing OWL ontologies in terms of the categories provided by the foundational ontology UFO as implemented in the gUFO OWL super-structure. We have tested these rules for consistency using the formal language Alloy and its model-finding computational support. The rules were then implemented in a free open-source software named Scior. We evidenced the efficiency of the inference rules and Scior in both open and closed-world assumption scenarios by testing them in an empirical evaluation. The experiment used diverse models from a catalog created by different modelers, for a wide range of domains and purposes, and having different quantities of classes. The evaluation results showed that, on average, Scior could automatically exclude at least five leaf classifications (from eight possibilities) with only 20% of the classes provided as seeds.

In the future, we intend to perform additional user studies. We are particularly interested in assessing Scior's ease of use and effectiveness in a human-in-the-loop interactive model. Further, we intend to study the application of Scior with ontologies outside the repository we have employed for testing here. This should rule out any potential effects

of recurrent patterns of taxonomic structures present in the repository. Finally, we will investigate how this rule-based approach can be combined with other ongoing initiatives in which some of us are involved and that employ Graph Neural Networks to address the grounding of ontologically neutral models into UFO foundational categories [25].

Acknowledgements

João Paulo A. Almeida is funded by the Brazilian agencies CNPq (313687/2020-0) and FAPES (281/2021 and 1022/2022).

References

[1] Farquhar A, Fikes R, Rice J. The ontolingua server: A tool for collaborative ontology construction. International journal of human-computer studies. 1997;46(6):707-27.
[2] Levesque HJ, Brachman RJ. Expressiveness and tractability in knowledge representation and reasoning. Computational intelligence. 1987;3(1):78-93.
[3] Guizzardi G, et al. Types and taxonomic structures in conceptual modeling: A novel ontological theory and engineering support. Data & Knowledge Engineering. 2021;134:101891.
[4] Nicola JRM. A formal analysis of Identity and Sortality in the Unified Foundational Ontology (UFO) [PhD thesis]. Federal University of Espírito Santo; 2021.
[5] Guizzardi G. The role of foundational ontologies for conceptual modeling and domain ontology representation. In: 7th Intl. Baltic Conf. on Databases and Information Systems; 2006. p. 17-25.
[6] Guizzardi G, et al. The Role of Foundational Ontologies for Domain Ontology Engineering: An Industrial Case Study in the Domain of Oil and Gas Exploration and Production. International Journal of Information System Modeling and Design. 2010;1(2):1-22.
[7] Guizzardi G. Ontology, Ontologies and the "I" of FAIR. Data Intelligence. 2020 01;2(1-2):181-91.
[8] Bittner T. Logical properties of foundational mereogeometrical relations in bio-ontologies. Applied Ontology. 2009;4(2):109-38.
[9] Flügel S, Glauer M, Neuhaus F, Hastings J. When One Logic is Not Enough: Integrating First-order Annotations in OWL Ontologies; 2022.
[10] Varzi A. Carnapian Engineering. In: Ontology Makes Sense; 2019. p. 3-23.
[11] Almeida JPA, et al. gUFO: A Lightweight Implementation of the Unified Foundational Ontology (UFO). online: https://purl.org/nemo/doc/gufo; 2020.
[12] Guizzardi G, et al. UFO: Unified Foundational Ontology. Applied Ontology. 2022;17(1):167-210.
[13] Guarino N, et al. An overview of OntoClean. In: Handbook on ontologies; 2009. p. 201-20.
[14] Borgo S, et al. DOLCE: A descriptive ontology for linguistic and cognitive engineering. Applied ontology. 2022;17(1):45-69.
[15] Loebe F, et al. GFO: The General Formal Ontology. Applied Ontology. 2022;17(1):1-36.
[16] Verdonck M, et al. Insights on the Use and Application of Ontology and Conceptual Modeling Languages in Ontology-Driven Conceptual Modeling. In: Conceptual Modeling. vol. 9974; 2016. p. 83-97.
[17] Jackson D. Alloy: A Lightweight Object Modelling Notation. ACM Transactions on Software Engineering and Methodology. 2002;11(2):256-90.
[18] Hustadt U. Do we need the closed world assumption in knowledge representation? In: Reasoning about Structured Objects: Knowledge Representation Meets Databases. vol. 1. CEUR-WS.org; 1994. .
[19] Barcelos PPF, et al. A FAIR Model Catalog for Ontology-Driven Conceptual Modeling Research. In: Conceptual Modeling. vol. 13607 of LNCS; 2022. p. 3-17.
[20] Krasnoperova AV. Representing real-world semantics in OWL ontologies [Master of Science in Business - MScB]. University of British Columbia; 2006.
[21] Keet CM, et al. Ontology Authoring with FORZA. In: 22nd ACM Intl. Conf. on Information & Knowledge Management; 2013. p. 569–578.
[22] Emeruem C, et al. BFO Classifier: Aligning Domain Ontologies to BFO. In: Joint Ontology Workshops 2022 Episode VIII: The Svear Sommar of Ontology. vol. 3249; 2022. .

[23] Völker J, et al. AEON–An approach to the automatic evaluation of ontologies. Applied Ontology. 2008;3(1-2):41-62.
[24] Guizzardi G. Ontological foundations for structural conceptual models. University of Twente; 2005.
[25] Ali SJ, et al. Enabling Representation Learning in Ontology-Driven Conceptual Modeling using Graph Neural Networks. In: 35th Intl. Conf. on Advanced Information Systems Engineering; 2023. .