An Extensible Approach to Multi-level Ontology Modelling

Hermann Bense[1][a] and Bernhard G. Humm[2][b]
[1]*bense.com GmbH, Schwarze-Brüder-Str. 1, 44137 Dortmund, Germany*
[2]*Hochschule Darmstadt - University of Applied Sciences, Haardtring 100, 64295 Darmstadt, Germany*

Keywords: Ontology Engineering, Multi-level Ontology Modelling, Class, Instance, Inheritance, Multi-facet Behaviour, RDF/RDFS, SPARQL.

Abstract: One major challenge in ontology engineering is deciding whether an entity should be modelled as a class or as an instance. Different modelling traditions and guidelines lead to different modelling decisions. This causes problems when integrating ontologies modelled according to different guidelines and trying to query over integrated ontologies. This article proposes a modelling approach which rigorously utilizes multi-level ontology modelling. In particular, multi-facet behaviour allows entities to be modelled as classes and instances simultaneously, where needed. It is argued that this supports simplicity, expressiveness, modularity, flexibility and extensibility of ontologies. The guidelines can be fully implemented using the W3C standards RDF/RDFS and SPARQL, allowing to implement inheritance behaviour using standardized inferencing mechanisms.

1 INTRODUCTION

One of the greatest challenges in ontology engineering is deciding whether something should be modelled as a class or as an instance. This is stated in the standard textbook 0 and we agree with the authors. They write “Since a semantic model must respond to competency questions coming from different stakeholders, it is quite possible that one work practice has a tradition of considering something to be a class, whereas another is accustomed to thinking of it as an instance” (Allemang et al., 2020, p. 434). They give the example of endangered species. For field zoologists who are tracking animals in wildlife, the species is a class whose members are the individual animals they are tracking. For the administrator in the federal agency that lists endangered species, the species is an instance to be put in such a list. The designer of a single model who wants to answer competency questions from both of these stakeholder communities is faced with a challenge.

Different modelling traditions and guidelines lead to different modelling decisions, modelling entities either as classes or as instances. This causes problems when integrating ontologies modelled according to different guidelines and trying to query over integrated ontologies.

For example, in the Vertebrate Taxonomy Ontology (VTO) (Phenoscape, 2021) and the NCBI Taxonomy (NCBI, 2021), all biological species are modelled as classes. For example, the species polar bear (ursus maritimus) is modelled as `owl:class`[1] in VTO:

```
<owl:Class rdf:about="http://
  purl.obolibrary.org/obo/VTO_0010492">
  <rdfs:label>Ursus maritimus</rdfs:label>
</owl:Class>
```

In contrast, in the Wildlife Ontology (WO) (BBC, 2021) and in the Global Biodiversity Information Facility (GBIF, 2021), biological species are modelled as instances. Polar bear (ursus maritimus) is modelled in GBIF under `gbifID 3032077871` as an instance of class `species`.

The authors of the WO state on (BBC, 2021): “One perennial problem associated with modelling biological taxonomies using RDF is whether to attempt to model individual species as classes, or whether to simply model species as instances of a

[a] https://orcid.org/0000-0002-2562-224X
[b] https://orcid.org/0000-0001-7805-1981
[1] All code examples are typeset using a typewriter font.

Bense, H. and Humm, B.
An Extensible Approach to Multi-level Ontology Modelling.
DOI: 10.5220/0010684200003064
In *Proceedings of the 13th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2021) - Volume 3: KMIS*, pages 184-193
ISBN: 978-989-758-533-3; ISSN: 2184-3228

generic Species class. The latter approach is simpler and avoids creating a huge ontology that attempts to model all biological organisms. Existing ontologies have taken different approaches to resolving this issue, some choosing one style, others another. At present there doesn't seem to be a consensus. With this in mind, the Wildlife Ontology adopts the simpler of the two approaches, i.e. modelling species as instances of a Species class, as this maximises interoperability with many of the existing Linked Data sources, particularly dbpedia, which adopt similar approaches."

We agree with the authors of the WO. For the purpose of annotating BBC wildlife documentaries, modelling biological species as instances is simple and sufficient. However, for use cases where individual animals have to be modelled, too, it is not sufficient to model species as instances only.

While in the four ontologies mentioned above, biological species are consistently modelled either as classes only or as instances only, in Wikidata (Wikimedia, 2021) as a crowd-sourced general-purpose ontology, there is a mixture of modelling biological species sometimes as classes and sometimes as instances. The species `dog`[2] is modelled as a class (subclass of `domesticated_mammal`[3]), with the concrete dog `Laika`[4] from the Soviet spacecraft Sputnik 2 as an instance of this class. On the other hand, the species `polar_bear`[5] is modelled as an instance of class `taxon`[6]. `taxon` is marked as a `second-order class`[7] which is an example of *multi-level modelling* (Neumayr et al. 2009; Almeida, et al., 2019; Brasileiro et al., 2016a; Brasileiro et al., 2016b; Carvalho and Almeida, 2018). Thus, the instance `polar bear` can also be considered a class. The concrete polar bear `Knut`[8] which was born in the Berlin Zoological Garden is modelled as an instance of this class.

The contribution of this paper is a modelling approach which rigorously utilizes *multi-facet* (Atkinson and Kühne, 2001; Neumayr et al., 2009; Frank, 2014) behaviour, an aspect of multi-level modelling, allowing entities to be modelled as classes (*class facet*) *and* instances (*instance facet*) simultaneously, where needed. This approach combines the advantages of both approaches, (a) modelling concepts like biological species as classes only and (b) modelling them as instances only, alleviating their disadvantages. It furthermore avoids inconsistencies that may arise when no clear modelling guidelines are specified. The multi-facet behaviour lifts the need for an either-or decision between class and instance and we regard it like cutting a Gordian knot in ontology engineering. In contrast to related work, our guidelines can be fully implemented using the W3C standards RDF/RDFS (W3C, 2014), and SPARQL (W3C, 2013a), allowing to implement inheritance behaviour using standardized inferencing mechanisms. It supports simplicity, expressiveness, modularity, flexibility and extensibility of ontologies. The importance of those characteristics we know from our extensive experience in ontology engineering and developing ontology-based semantic applications for different application domains, including life sciences, tourism, industrial manufacturing, energy, robot journalism, and culture.

The remainder of this article is structured as follows. In Section 2 we discuss related work. Section 3 presents an example ontology which we use throughout this paper. In Sections 4-7, we present the multi-facet behaviour and its implications on property types, inheritance, querying and inferencing. Section 8 discusses the approach. Section 9 concludes the paper and indicates future work.

2 RELATED WORK

In the RDFS specification (W3C, 2014), classes are described as groups of resources and instances as their members. Alemang et al. complain that a common reaction to the difficult distinction between classes and instances is simply to define everything as a class. They grade this as an antipattern, i.e., a bad modelling practice, which they call "rampant classism" (0, 2020. p.436). We agree with them.

(Noy and McGuinness, 2021) is one of the earlier guidelines for ontology engineering. A criterium they define for the decision to model something as a class or an instance is the lowest level of granularity in the representation. They see individual instances as the most specific concepts represented in a knowledge base. Then individual instances could be transferred into a set of classes, if they form a natural hierarchy.

In the field of multi-level ontology modelling, various authors recommend modelling pairs of

[2] https://www.wikidata.org/wiki/Q144
[3] https://www.wikidata.org/wiki/Q57814795
[4] https://www.wikidata.org/wiki/Q53662
[5] https://www.wikidata.org/wiki/Q33609 (All Wikidata examples accessed 2021-03-01)
[6] https://www.wikidata.org/wiki/Q16521
[7] https://www.wikidata.org/wiki/Q24017414
[8] https://www.wikidata.org/wiki/Q159697

classes where a class is associated with a corresponding type class e.g. `(Car, Car_Model)` (Atkinson and Kühne, 2001; Pirotte et al., 1994; Odell, 1994; Guizzardi et al., 2015; Almeida et al., 2019; Brasileiro et al., 2016a; Brasileiro et al., 2016b; Carvalho and Almeida, 2018; Pan and Horrocks, 2001). The modelling pattern is called *materialization pattern* in (Pirotte et al., 1994); The term *powertype* is used in (Odell, 1994) and (Guizzardi, et al., 2015). In the example, `Car_Model` is the powertype of `Car`. Instances of the powertype `Car_Model` like `2CV_Model` have properties like `maxSpeed` or `Year_of_Design`. `2CV` is a subclass of the class `Car`, corresponding to `2CV_Model`. Its instances like `My_2CV` have properties like `Number_Plate` or `Milage`. With conventional modelling approaches it is not possible to mix the properties of base class and its powertype into one class, since it makes no sense to have a property `Number_Plate` for a `Car_Model`. Our approach is simpler insofar as it does not require separate entities `Car` and `Car Model` while still explicitly distinguishing between the different facets.

Another approach to avoid separate entities `Car` and `Car Model` is described in (Neumayr et al. 2009), called *Multi-Level Domain Modeling*. Properties are annotated with meta attributes which define rules for inheritance and instantiation. E.g. for the base class `Car` the data property `maxSpeed` is annotated with `:model` to refer to a `Car_Model` while the data property `Milage` is annotated with `:physical_entity` to refer to an individual car. This allows merging the properties of `Car` and `Car_Model` into only one class `Car` while keeping the semantics of the properties clear.

In a comparable approach (Frank, 2014), level numbers are assigned to properties. For `Car` the property `Milage` would be defined with `level=0` which corresponds to the `:physical_entity` annotation in (Neumayr et al. 2009), while `maxSpeed` would be annotated with `level=1` which corresponds to the `:model` annotation. Annotating properties with multiple levels or with different levels for different contexts is not possible, as is enabled in our approach.

In (Brasileiro et al., 2016a), a fixed separation of entities into individuals, 1st order type, 2nd order type and 3rd order type is proposed. Our approach is more flexible as it allows context-dependent viewpoints. So, polar bear may be an individual in one use case context (tagging wildlife series) but a 1st order type in another context (tracking concrete animals). The facets of entities could also evolve during the development time of an ontology. From our experience in ontology modelling and developing ontology-based semantic applications this is important.

Multi-facet behaviour as used in (Neumayr et al., 2009) may regard entities as classes and instances simultaneously, where needed. In contrast, OWL 1 DL requires a strict separation between the classes and individuals (W3C, 2012). Insofar, multi-facet behaviour is not compatible with OWL1 DL. However, OWL 2 DL relaxes this separation somewhat to allow different uses of the same term which is called *Punning*. In (W3C, 2012, Section 2.4.1), the following example is given: “e.g., Eagle, to be used for both a class, the class of all Eagles, and an individual, the individual representing the species Eagle belonging to the (meta)class of all plant and animal species”. However, OWL 2 DL still imposes certain restrictions, e.g., a name can only be used for one kind of property (W3C, 2012).

In the next sections, we present an extensible modelling approach rigorously using multi-facet behaviour and demonstrate it by means of a sample ontology.

3 EXAMPLE: ANIMAL ONTOLOGY

In this article, we use a small ontology as an illustrating example in which we modelled some biological species. We call this sample ontology *Animal Ontology (AO)*. The AO is based on the WO (BBC, 2021)[9]. The basis of AO is the taxonomy established in biology classifying living entities using the taxons kingdom, phylum, class[10], order, family, genus, and species.

Like the authors of WO, we favour a simple modelling approach modelling concrete species like polar bear as instances. See the following example in RDF Turtle syntax.

```
@prefix : <http://h-da.de/animalontology/>.
@prefix rdf: <http://
  www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://
  www.w3.org/2000/01/rdf-schema#>.
```

[9] We made only minor changes, e.g., the use of SKOS.

[10] The biological taxon “class” is not to be confused with the ontology concept of a class. We distinguish both by using different namespaces.

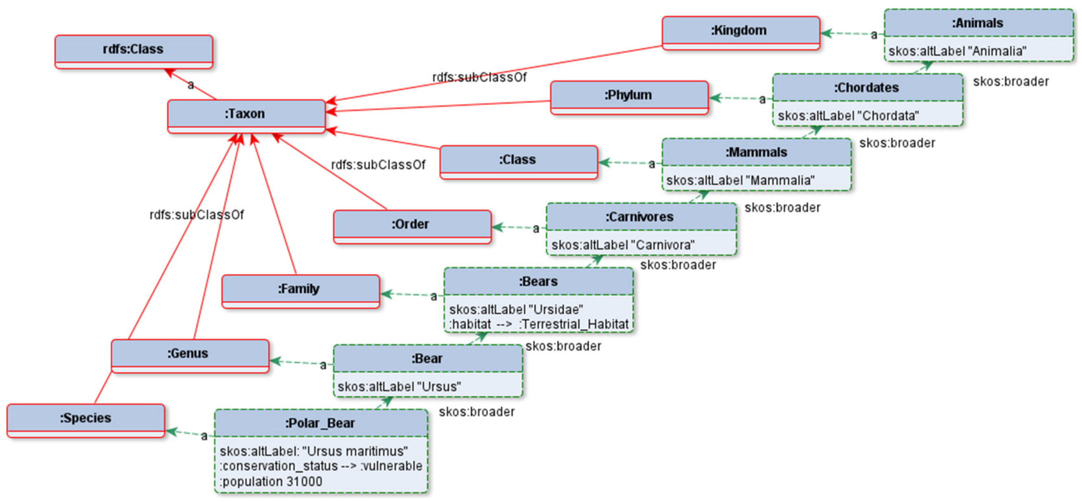

Figure 1: Subset of the Animal Ontology (AO).

```
@prefix skos: <http://
  www.w3.org/2004/02/skos/core#>.
:Species a rdfs:Class ;
  skos:prefLabel "Species" ;
  rdfs:subClassOf :Taxon .
:Polar_Bear a :Species ;
  skos:prefLabel "Polar Bear" ;
  skos:altLabel "Ursus maritimus" ;
  skos:broader :Bear .
```

See Fig. 1 for a visualization of a subset of AO with one example for each taxon. Classes are depicted with red solid lines and instances with green dashed lines[11].

In the AO, `:Taxon` is an `rdfs:Class` with subclasses `:Kingdom`, `:Phylum`, `:Class`, `:Order`, `:Family`, `:Genus`, and `:Species`. `:Polar_Bear` is an instance of `:Species`, `:Bear` an instance of `:Genus`, `:Animals` an instance of `:Kingdom` etc. We use the object property `skos:broader` to specify the biological taxonomy, e.g.,

```
:Polar_Bear skos:broader :Bear.
:Chordates skos:broader :Animals.
```

In addition, concrete taxons are attributed with properties, e.g.,

```
:Polar_Bear skos:altLabel "Ursus maritimus".
```

The WO was modelled for the use case of annotating BBC wildlife documentaries. The AO is based on the WO and could well serve a similar use case. But what if the use cases of AO shall be extended, e.g., if field zoologists want to model concrete animals that they track or the administration of a zoo wants to manage their animals?

The simple modelling approach of AO presented above is not capable of supporting such use cases since species like `:Polar_Bear` are not modelled as a class which can be instantiated with concrete animals like `:Knut`.

In the next section we demonstrate how the simple modelling approach can be extended to support new use cases using multi-facet behaviour.

4 MULTI-FACET BEHAVIOUR

RDF and RDFS provide means for specifying classes and instances. A class is specified as an instance of `rdfs:Class` (or any of its subclasses like `owl:Class`), using the object property `rdf:type` (abbreviated in Turtle with `a`), e.g.,

```
:Species a rdfs:Class.
```

An instance is specified using a `rdf:type` relationship to a class, e.g.,

```
:Polar_Bear a :Species.
```

However, the standards provide no restrictions on the usage of classes and instances 0. It is not required to use classes and instances whatsoever, i.e., not using `rdf:type` at all is allowed. It is allowed to model classes with several superclasses (`rdfs:subClassOf` - multiple inheritance). It is allowed to model something to be instance of more than one class, which is not supported in object-oriented languages like Smalltalk, Java, C# or C++.

[11] To keep the figure clear we omitted additional green boxes for the target of object properties like `conservation_staus :vulnerable` but instead indicated this with a dashed arrow -->

In RDF/RDFS it is not forbidden to specify an entity as an instance of something which has not been specified as a class. Although this is syntactically correct, we consider this a modelling flaw.

It is also allowed to model something as a class and an instance simultaneously. Every class is an example of this pattern since every class is an instance of `rdfs:Class`. Also `rdfs:Class` is an example of this since it is a class and additionally an instance of itself.

In (Atkinson and Kühne, 2001) and (Neumayr et al., 2009) this is called *multi-facet.* It is argued that this may considerably reduce unnecessary complexity in ontologies with multiple levels. We agree with them and use the multi-facet behaviour to add capabilities to ontologies like AO incrementally when required by new use cases.

For example, when modelling concrete animals in a zoo like the polar bear `:Knut`, then the concept `:Polar_Bear`, modelled as an instance of class `:Species`, can additionally be specified as a class:

```
:Polar_Bear a :Species , rdfs:Class.
```

The concrete polar bear `:Knut` can now be modelled as follows.

```
:Knut a :Polar_Bear;
  skos:prefLabel "Knut";
  :sex "male";
  :birthDate "2006-12-05"^^xsd:date;
  :birthPlace :Berlin_Zoological_Garden;
  :deathDate "2011-03-19"^^xsd:date;
  :deathPlace :Berlin_Zoological_Garden;
  rdfs:seeAlso <https://en.wikipedia.org/
    wiki/Knut_(polar_bear)>.
```

See Fig. 2.

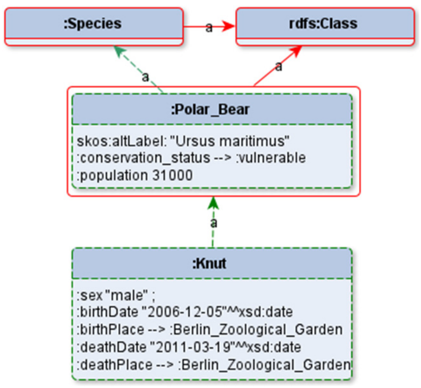
:Species
a
rdfs:Class
a
a
:Polar_Bear
skos:altLabel: "Ursus maritimus"
:conservation_status --> :vulnerable
:population 31000
a
:Knut
:sex "male" ;
:birthDate "2006-12-05"^^xsd:date
:birthPlace --> :Berlin_Zoological_Garden
:deathDate "2011-03-19"^^xsd:date
:deathPlace --> :Berlin_Zoological_Garden

Figure 2: Modelling `:Polar_Bear` as an instance and as a class simultaneously.

The multi-facet behaviour of taxon `:PolarBear` can be seen by the green dashed line indicating that `:PolarBear` is an instance of class `:Species` (instance facet) and the red solid line indicating that `:Polar_Bear` is a `rdfs:Class` with `:Knut` as an instance (class facet).

5 PROPERTY TYPES

Properties are used for specifying information about entities like `:Polar_Bear` or `:Knut`. Datatype properties like `:birthDate` or `:population` relate to literal values like strings, numbers or dates, e.g.

```
:Polar_Bear :population 31000.
:Knut :birthDate "2006-12-05"^^xsd:date.
```

Object properties like conservation_status relate to other entities, e.g.

```
:Polar_Bear :conservation_status
  :vulnerable.
:vulnerable a :Conservation_status.
:Conservation_status a rdfs:Class.
```

Datatype properties as well as object properties always relate to the instance facet of an entity, indicated by green dashed lines in Fig. 1 and 2. In the example above, the property `:birth_date` refers to the instance `:Knut`.

In the other example, `:conservation` status refers to the instance facet of `:Polar_Bear` , not to the class facet: the species polar bear is vulnerable[12]. Does the fact that the species polar bear is vulnerable mean that the individual polar bear Knut is vulnerable? No: Knut may happily live in the Berlin Zoo, while his species as a whole is vulnerable. The same applies for the population which refers to the species polar bear, but not to the instance Knut.

Let us look at another example of an object property: habitat.

```
   :Bears :habitat :Terrestrial_habitat.
   :Terrestrial_habitat a :Habitat.
   :Habitat a rdfs:Class.
```

These RDF triples express that bears live on terrestrial habitat, i.e., that they are land animals. Again, this information relates to the instance facet of the biological family of bears. Does this information also apply to all genus and species of the family bears? Yes, indeed it can be implied that polar bears, American black bears, Panda bears etc. all are land

[12] According to ICUN https://www.iucn.org/ vulnerable is the status next to endangered.

animals. And how about individual animals like Knut? Yes, indeed it can be implied that all individual polar bears, American black bears, Panda bears etc., including Knut, are land animals.

It is obvious that, although both properties `:conservation_status` and `:habitat` relate to the instance facet, they are semantically different. The difference is how information can be inferred between the facets along hierarchies within the facets. Whereas information about the habitat can be inferred from broader concepts to narrower concepts in the biological taxonomy and even to instances, this is not the case for information about the population or the conservation status.

Please note that this semantic difference is independent of the distinction between object properties and datatype properties. If, e.g., `:conservation_status` had been modelled as a datatype property (`:Polar_Bear :conservation_status "vulnerable"`) then the intended inference would still be the same.

6 INHERITANCE

In the field of object-orientation, inferring information along subclass-hierarchies is called *inheritance* (Nierstrasz, 1989). We borrow this term for inferring information between facets as well as along hierarchies within facets, e.g., `rdfs:subClassOf` hierarchies within the class facet or `skos:broader` hierarchies within the instance facet.

In our example, the hierarchy between taxons is specified by the object property `skos:broader` .

```
:Polar_Bear a :Species; skos:broader :Bear.
:Bear a :Genus; skos:broader :Bears.
:Bears a :Family; skos:broader :Carnivores.
:Carnivores a :Order; skos:broader :Mammals.
:Mammals a :Class; skos:broader :Chordates.
:Chordates a :Phylum; skos:broader :Animals.
:Animals a :Kingdom.
```

Let us now reconsider the sample properties `:birth_date`, `:population`, `:conservation_status` and `:habitat`.

1. `:birth_date "2006-12-05"` refers to an instance of a concrete species, e.g., `:Knut`, only.
2. `:population 31000` refers to a concrete species, genus, family etc., e.g., `:polar_bear`, but cannot be inherited from broader to narrower taxons, and not to their instances.
3. `:conservation_status : vulnerable` refers to a concrete taxon, e.g., the species `:polar_bear`. If a conservation status would be specified for an entire genus or family then it could be inferred that all species belonging to this genus or family have the same conservation status, hence can be inherited along the `skos:broader` hierarchy. However, it cannot be inherited to instances of concrete species like `:Knut`.
4. `:habitat :Terrestrial_habitat` refers to a concrete species, genus, family etc., e.g., `:Bears`, it can be inherited along the `skos:broader` hierarchy, e.g., from `:Bears` to `:Bear` and from `:Bear` to `:Polar_bear`. Additionally, it can be inherited from concrete species like `:Polar_Bear` to instances of this species like `:Knut`.

In order to use inferencing for implementing inheritance behaviour, it is necessary to explicate the intended behaviour of properties like `:birth_date`, `:population`, `:conservation_status`, and `:habitat`. Consider the following declarations.

```
:birth_date a rdfs:Property.
:population a rdfs:Property.
:conservation_status a rdfs:Property,
  :BroaderInheritedProperty.
:habitat a rdfs:Property,
  :BroaderInheritedProperty,
  :TypeInheritedProperty
```

Those RDF triples express that all four properties are of type `rdfs:Property`. `:birth_date` and `:population` have no intended inheritance behaviour. `:conservation_status` and `:habitat` are of type `:BroaderInheritedProperty` in order to express that those properties can be inherited along the `skos:broader` hierarchy from broader to narrower taxons. Additionally, `:habitat` is of type `:TypeInheritedProperty` in order to express that this property can additionally be inherited along the `rdf:type` relationship from classes to their instances.

Please note that individual properties can be of any of all four possible combinations:

1. Of both types `:BroaderInheritedProperty` and `:TypeInheritedProperty`, e.g., `:habitat`;
2. Only `:BroaderInheritedProperty`, e.g., `:conservation_status`;

3. Only `TypeInheritedProperty`;
4. None of both, e.g. `:birth_date`.

Furthermore, inheritance behaviour may apply to object properties and datatype properties alike.

In the next section we show that such property declarations can be used for implementing inheritance via inferencing.

7 QUERYING AND INFERENCING

Ontology engineering is no end in itself but has the purpose of using information modelled. Query languages allow accessing information in ontologies. SPARQL (W3C, 2013a) is the standardized query language for RDF. We use SPARQL for demonstrating examples.

The following SPARQL query selects all information about `:Knut`.

```
SELECT *
WHERE{:Knut ?p ?o.}
```

This query will return for RDF triples with subject `:Knut` all predicates `?p` and objects `?o` explicitly specified, e.g., `:sex "male"`. However, it will not contain any information about Knut's habitat.

Likewise, a similar query for all information about the species polar bear will return all predicates and objects explicitly specified for the subject `:Polar_Bear` like `:population 31000`, but no information about the habitat of polar bears.

This is because habitat information is specified for a broader taxon, here the family of bears.

Inferencing can be used to implement inheritance behaviour as outlined in the previous section. We demonstrate this using SPARQL update (W3C, 2013b) for sample rules. Consider the following SPARQL `INSERT` statement specifying the rule for inheriting properties along `skos:broader` hierarchies.

```
INSERT {?s ?p ?o.}
WHERE  {?p a :BroaderInheritedProperty.
        ?s skos:broader* ?b.
        ?b ?p ?o.}
```

This rule applies for RDF triples with predicates `?p` that have been specified of type `:BroaderInheritedProperty` like, e.g., `:conservation_status` or `:habitat`. If a subject `?s` is in a transitive `skos:broader` relationship to some other subject `?b` then all triples with predicate `?p` and object `?o` can be inherited for subject `?s` .

Executing this SPARQL `INSERT` statement materializes the inferencing results in the ontology. Executing a SPARQL query about all information about the species `:Polar_Bear` will now, in addition to explicitly modelled information, also return inferred information
`:habitat :Terrestrial_habitat.`

However, a query for all information about `:Knut` will still not render habitat information because it is neither specified explicitly nor cannot it be inferred with the rule above. Therefore, now consider the following SPARQL `INSERT` statement specifying the rule for inheriting properties from classes to instances.

```
INSERT {?s ?p ?o.}
WHERE  {?p a :TypeInheritedProperty.
        ?s a ?c.
        ?c ?p ?o.}
```

This rule applies for RDF triples with predicates `?p` that have been specified of type `:TypeInheritedProperty` like, e.g., `:habitat`. If a subject `?s` is in a `rdf:type` relationship to some class `?c` then all triples with predicate `?p` and object `?o` can be inherited for subject `?s` .

After executing this SPARQL `INSERT` statement, querying for all information about Knut will now additionally render
`:habitat :Terrestrial_habitat.`
It shall be noted that in practice, all rules have to be executed repeatedly until no rule fires any more, as is common in forward-chaining rule-based systems.
We would also like to add a remark on our use of `skos:broader` in the example ontology AO. `skos:broader` is an informal property for expressing hierarchical relations including whole-part, spatial, temporal, etc. There may be good reasons for ontology engineers for preferring such an informal relation over a more formally specified relation like `rdfs:subClassOf` in a certain context. Re-using existing ontologies for new use case contexts is common practice in ontology engineering and in developing ontology-based semantic applications. We have deliberately chosen an informal property like `skos:broader` for the AO in order to demonstrate how semantics for different use case contexts may be added incrementally, independently of the modelling style of the initial ontology. If a more formally specified relationship like `rdfs:subClassOf` is deemed necessary in a certain use case context then a SPARQL rule for inferring `rdfs:subClassOf` relationships from `skos:broader` relationships may easily be added. This is the basis for the flexibility and extendibility of our approach which we discuss in the next section.

8 DISCUSSION

"Everything should be made as simple as possible, but no simpler" is a quote that has been attributed to Albert Einstein (although he presumably never said it literally) (Championing Science, 2021). In software engineering, the acronym KISS ("keep it simple, stupid") has been used to express the same principle of *simplicity*. Also, for ontology engineering it is a good practice to model entities as simple as possible.

In this article we suggest to model entities as classes only if this is actually needed, i.e., if in the domain of interest there are instances of this entity that need to be modelled. However, the addition "but no simpler" indicates that the *expressiveness* of the model must be adequate for the domain of interest, i.e., everything that is relevant can actually be modelled. The requirements for an ontology may vary between different usage contexts and over time. In the example of the AO, in one usage context it may be sufficient to model species as instances, e.g., for tagging wildlife documentaries. In another context it may be necessary to model concrete species like polar bear as classes, e.g., for documenting individual animals watched in wildlife. It may be the case that different subsets of an ontology shall be used for different contexts - this requires *modularity* of the ontology. It may also be the case that the usage context of an ontology evolves over time - this requires *extensibility* of the ontology.

We argue that rigorously utilizing the multi-facet behaviour for modelling ontologies supports simplicity, expressiveness, modularity, extensibility and flexibility. It does not enforce an either-or decision of modelling entities as classes or instances. Instead, it allows starting simple (simplicity, e.g., modelling `:Polar_bear` as an instance of class `:Species`) and extending as needed (adequate expressiveness and extensibility, e.g., modelling `:Polar_bear` additionally as a class). The instance facet (e.g., `:Polar_bear` as an instance) can be separated from the class facet (e.g., `:Polar_bear` as a class including its instances), supporting modularization via different ontology files.

Conciseness is related to simplicity, aiming at modelling entities with as few statements (e.g., RDF triples) as possible. Utilizing the multi-facet behaviour is more concise than using the materialization pattern. For example, modelling the AO with the materialization pattern would require modelling taxons and animals as separate entities. This impedes readability of the ontology and complicates the implementation of queries.

Lack of redundancy is related to conciseness. We have shown that using inferencing for implementing inheritance, redundancy in ontologies can largely be reduced.

Interoperability is important for linking ontologies. In our examples, we use standards like RDF/RDFS and SPARQL that support interoperability on a syntactic level. However, in our opinion, interoperability on the modelling style is as important. In the introduction we refer to four different animal ontologies, two of them modelling all species as classes (VTO and NCBI Taxonomy), the other two modelling them as instances (WO and GBIF). Utilizing multi-facet behaviour requires no either-or decision on class or instance. Instead, it allows combining the strengths of both modelling styles while alleviating their disadvantages. However, as stated in (W3C, 2012), multi-facet behaviour could impede OWL DL reasoning.

In contrast to (Neumayr et al., 2009) and (Frank, 2014), we use standard modelling and query languages (RDF/ RDFS, SPARQL). In (Neumayr et al., 2009), SQL must be extended for querying M-Objects; In (Frank, 2014), queries are not discussed at all.

In (Neumayr et al., 2009), inheritance behaviour is not discussed. In (Frank, 2014), inheritance is restricted to the definition of properties, i.e., using a property specified in a higher-level object in a lower level. Inferring property values between facets and along hierarchies within facets, e.g., `rdfs:subClassOf` or `skos:broader`, is not discussed.

Our approach allows a large level of *flexibility*, no "one size fits all". Depending on the application context entities can be assigned different modelling levels, properties can be annotated with different inheritance behaviour and different inheritance rules can be implemented. This extends the flexibility of existing multi-level modelling approaches.

9 CONCLUSIONS AND FUTURE WORK

In this paper we discuss a major challenge in ontology engineering, namely deciding whether an entity should be modelled as a class or as an instance. Different modelling traditions and guidelines lead to different modelling decisions, enforcing an either-or decision.

In contrast, we propose to rigorously utilize multi-facet behaviour allowing entities to be modelled as classes *and* instances simultaneously, where needed. We argue that it supports simplicity, expressiveness, modularity, flexibility and extensibility of ontologies. Furthermore, we present means for implementing inheritance behaviour between facets and along hierarchies within facets by using inferencing. We argue that this reduces redundancy and increases conciseness.

We have applied the modelling approach presented in this paper to concrete domain-specific ontologies in practice and we continue to do so. Currently, we are working on an ontology for machine learning (ML) (Humm and Zender, 2021; (Humm et al., 2021), modelling ML approaches, ML libraries, their characteristics and interrelationships, and more. This ontology shall be used in various use case contexts, including teaching ML, supporting ML engineers when designing ML applications, supporting automated orchestration of ML applications (AutoML) or chat bots for answering questions about ML. The flexibility of our approach can be particularly useful in using the ontology in those different use case contexts.

REFERENCES

Dean Allemang, Jim Hendler, Fabien Gandon (2020). Semantic Web for the Working Ontologist - Effective Modeling in RDFS and OWL, Third Edition. ACM Books series, Nbr. 33, 2020, ISBN 978-1-4503-7614-3

Phenoscape (2021): Vertebrate Taxonomy Ontology (VTO). Online https://github.com/phenoscape/vertebrate-taxonomy-ontology accessed 2021-04-21.

National Center for Biotechnology Information (NCBI) (2021): NCBI Taxonomy. Online https://www.ncbi.nlm.nih.gov/taxonomy, accessed 2021-04-21.

British Broadcasting Corporation (BBC) (2021): Wildlife Ontology (WO). Online https://www.bbc.co.uk/ontologies/wo, accessed 2021-04-21.

Global Biodiversity Information Facility (GBIF) (2021). Online https://www.gbif.org, accessed 2021-04-21.

Wikimedia (2021): Wikidata. Online https://www.wikidata.org, accessed 2021-04-21.

Colin Atkinson, Thomas Kühne (2001). The Essence of Multilevel Metamodeling. In: Gogolla M., Kobryn C. (eds) ≪ UML ≫ 2001 − The Unified Modeling Language. Modeling Languages, Concepts, and Tools. UML 2001. Lecture Notes in Computer Science, vol. 2185. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45441-1_3.

Bernd Neumayr, Katharina Grün, Michael Schrefl (2009). Multi-Level Domain Modeling with M-Objects and M-Relationships. Sixth Asia-Pacific Conference on Conceptual Modelling (APCCM 2009), Wellington, New Zealand, January 20-23 2009. CRPIT 96, Australian Computer Society 2009, ISBN 978-1-920682-77-4.

Ulrich Frank (2014). Toward a New Paradigm of Conceptual Modeling and Information Systems Design. Business & Information Systems Engineering 6(6), Dec. 2014, pp. 319-337. DOI: 10.1007/s12599-014-0350-4

World Wide Web Consortium (W3C) (2014): RDF Schema 1.1, W3C Recommendation 25 February 2014. Online https://www.w3.org/TR/rdf-schema, accessed 2021-04-21.

World Wide Web Consortium (W3C) (2013a): SPARQL 1.1 Query Language W3C Recommendation 21 March 2013. Online https://www.w3.org/TR/sparql11-query, accessed 2021-04-21.

World Wide Web Consortium (W3C) (2013b): SPARQL 1.1 Update, W3C Recommendation 21 March 2013. Online https://www.w3.org/TR/sparql11-update, accessed 2021-04-21.

Natalya F. Noy, Deborah L. McGuinness (2021). Ontology Development 101: A Guide to Creating YourFirst Ontology. Technical reports, Stanford University, Stanford, CA, 94305, Online https://protege.stanford.edu/publications/ontology_development/ontology101.pdf , accessed 2021-08-03.

Alain Pirotte, Esteban Zimanyi, David Assart, Tatiana Yakusheva (1994). Materialization: a powerful and ubiquitous abstraction pattern. VLDB, Morgan Kaufmann, 1994, pp. 630-641.

James J. Odell (1994). Power Types. Journal of Object-Oriented Programming, Volume 7(2), 1994 , pp. 8-12.

Giancarlo Guizzardi, Joao Paulo A. Almeida, Nicola Guarino, Victorio A. Carvalho (2015). Towards an Ontological Analysis of Powertypes. International Workshop on Formal Ontologies for Artificial Intelligence (FOFAI), 2015.

Joao Paulo A. Almeida, Victorio A. Carvalho, Freddy Brasileiro, Claudenir M. Fonseca, Giancarlo Guizzardi (2019). Multi-Level Conceptual Modeling: Theory and Applications. 2019.

Freddy Brasileiro, Joao Paulo A. Almeida, Victorio A. Carvalho, Giancarlo Guizzardi (2016a). Expressive Multi-Level Modeling for the Semantic Web. 15th International Semantic Web Conference (ISWC 2016). Lecture Notes in Computer Science October 2016, DOI: 10.1007/978-3-319-46523-4_4.

Freddy Brasileiro, Joao Paulo A. Almeida, Victorio A. Carvalho, Giancarlo Guizzardi (2016b). Applying a Multi-Level Modeling Theory to Assess Taxonomic Hierarchies in Wikidata. Wiki Workshop at 25th Int. Conference Companion World Wide Web, 2016, pp. 975-980.

Victorio A. Carvalho, Joao Paulo A. Almeida (2018). Toward a Well-Founded Theory for Multi-Level Conceptual Modeling. Software and Systems Modeling 17(1), Springer Verlag, February 2018. DOI: 10.1007/s10270-016-0538-9.

Jeff Z. Pan, Ian Horrocks (2001). Metamodeling Architecture of Web Ontology Languages. Proc. of the

2001 International Semantic Web Working Symposium, 2001, pp. 131-149.

World Wide Web Consortium (W3C) (2012): OWL 2 Web Ontology Language, New Features and Rationale (Second Edition), W3C Recommendation 11 December 2012. Online https://www.w3.org/TR/owl2-new-features, accessed 2021-04-27.

Bernhard G. Humm, Alexander Zender (2021): An Ontology-Based Concept for Meta AutoML. Proceedings of the 17th International Conference on Artificial Intelligence Applications and Innovations (AIAI 2021), Crete, Greece, 25 – 27 June, 2021.

Bernhard G. Humm, Hermann Bense, Michael Fuchs, Benjamin Gernhardt, Matthias Hemmje, Thomas Hoppe, Lukas Kaupp, Sebastian Lothary, Kai-Uwe Schäfer, Bernhard Thull, Tobias Vogel and Rigo Wenning (2021): Machine intelligence today: applications, methodology, and technology. Hauptbeitrag Informatik Spektrum 44, 104–114, Springer Verlag Heidelberg, Germany, 2021. https://doi.org/10.1007/s00287-021-01343-1.

Oscar Nierstrasz (1989): A Survey of Object-Oriented Concepts. In Object-Oriented Concepts, Databases and Applications, ed. W. Kim and F. Lochovsky, pp. 3-21, ACM Press and AddisonWesley, 1989.

Championing Science (2021): In honor of Albert Einstein's birthday – Everything should be made as simple as possible, but no simpler. Online https://www.championingscience.com/2019/03/15/everything-should-be-made-as-simple-as-possible-but-no-simpler. accessed 2021-04-21.