Towards a General Ontology Theory

Hermann Bense*

bense.com Digital Publishing Company GmbH, Schwarze-Brüder-Str. 1,
44137 Dortmund, GERMANY, email: hb@bense.com

Abstract. Purpose: Since the beginning of the 2000s, numerous domain, upper level, and top level ontologies have been developed. In order to represent a model of the reality they contain tens to several thousand concepts, properties and axioms. These are instantiated into huge collections of facts about the world. In this paper, we propose a General Ontology Theory which focuses on a general formal description of ontological concepts. **Methodology**: We capture the fundamental structures and principles that that apply to the the elements of top level and upper level ontologies, potentially addressing overarching functionalities such as identity, abstraction, aggregation, classification, inheritance, subsumption, instantiation, and their relationships. To this end we use mathematical set theory to synthesize definitions and axioms along with guidelines for their application. **Findings**: The O4Top Ontology Blueprint is a schema that provides a foundation for conceptual and meta-level modeling. It supports the formalization of structural hierarchies for classes, relationship types and attributes. From that we can derive how particulars of classes are related to the set of their elements. As a result we are able to formulate subsumption axioms for classes and relationship types. We also can derive the hierarchical ordering merely from the set of particulars and their attributes. **Originality**: We define a minimal set of ontological elements that is sufficient for conceptual modeling in general. This supports the streamlining of ontologies and their terminologies.

Keywords: General Ontology Theory, Formal Concept Analysis, Instantiation, Intension, Extension, Principal Ideal, Principal Filter, Subsumption, Top Level Ontology

1 Introduction

To support the development and use of ontologies, both, top-level ontologies such as BFO, GFO, DOLCE, and UFO, and upper ontologies such as schema.org, and SUMO have been developed for more than two decades. Top-level ontologies assist modelers in the development of domain ontologies such as the GeneOntology[1]. Upper level ontologies serve as frameworks that modelers can use directly or extend as needed.

* https://orcid.org/0000-0002-2562-224X

[1] https://geneontology.org/

Both types of ontologies contain tens to thousands of concepts, properties and axioms to model reality. Top-level ontologies provide more abstract concepts such as *action, achievement, activity, amount of matter, agent, dimension, entity, event, goal, instrument, location, matter, occurrent, part, path, perdurant, process, quality, quantity, result, role, space, region, time, theme,* and so on. Upper level ontologies provide more concrete concepts such as *animal, article, artwork, body part, body of water, building, city, competition, contract, country, corporation, device, document, food, game, hospital, marriage, nuclide, person, plant, product, room, school, surgery, team, work*, and so on. Driven in particular by the developments in the area of the Semantic Web, there has been a great deal of fundamental research into the methodology of ontology development. The works of Noy and McGuiness [1], Arp et al. [2], and Allemang et al. [3] are representative for that. This was accompanied by the development of comprehensive axiom systems as presented by Masolo et al. in [4] by Guizzardi et al. in [5], and by Selway et al. [6]. We consider such comprehensive axiom systems to be Special Ontology Theories (SOT). In physics and chemistry there are essentially only a few building blocks such as atoms, neutrons, protons and electrons, from which complex molecules can be assembled. The interaction of these building blocks is governed by the laws of nature. By analogy, the question arises as to whether we can find a similarly small set of building blocks to create ontologies and what rules govern the interaction of their building blocks. Therefore, we focus on the design of a *General Ontology Theory* (GOT) that identifies a minimal set of ontology building blocks and a set of definitions and axioms to describe the laws of their interaction. The goal is to capture the fundamental structures and principles that hold across top-level, upper level, and domain ontologies, potentially addressing overarching functionalities such as *abstraction, aggregation, classification, inheritance, subsumption, instantiation, and identity.* Although there is already a lot of preliminary work in this field of research, there is no mathematically formalized axiom system that describes the laws between the basic building blocks of ontologies. Here we want to close this gap.

Our research approach is based on the paper of Nunamaker et al. [7] and that of Hevner et al. [8] which discusses 'Design Science in Information Systems Research'. The central research question is: What might be a minimal set of ontology building blocks that would form the basis for expressive conceptual and meta-level modeling? This leads to further questions: How can we use mathematical set theory to express the principle of ordering relations of classes and relationship types, and what

are the resulting regularities? Finally, what is the relationship between extension and intension of classes and relationship types?

The rest of the paper is organized as follows. In the Related Works section 2, we discuss related work in the areas of top-level ontologies and their axiomatic systems. In the Foundations section 3, we discuss naming conventions, non-strictly ordered sets, and the ordering relations for defining structures of classes, relationship types and data properties. We also define rules for instantiating data and object properties. The results is our minimal *ontology blueprint*. The Axioms section 4 derives *subsumption axioms* based on the *intension*[2] and *extension* of universals. In the Discussion section 6, we compare our methodology with other minimal top ontologies. In the Conclusion section 7 we summarize the results.

2 Related Work

In this section we describe selected issues from different areas of ontology engineering, that we consider important for a theory about ontologies. We will return to some of these issues in the Discussion section 6. Guarino discusses in [9] "A Minimal Ontology of Universals". We will propose our own ontology of universals and compare it to Guarino's. Suchanek et al. present in [10] with YAGO "a large ontology with high coverage and precision. ... YAGO is based on a clean logical model with a decidable consistency". To this end, they provide a rewriting system with a small set of rules. The rules are based on the relations `type`, `subClassOf`, `domain`, `range` and `subRelationOf`, and the classes `entity`, `class`, and `relation`.

Guizzardi et al. discuss in [5] a large set of axioms for formalizing UFO in first-order modal logic. In comparison, we will investigate how properties of universals can be used to define additional axioms for the subsumption of classes and relationship types. Herre et al. ([11], page 12) discuss axioms for their Abstract Top Level (ATO) of GFO. We will use these axioms implicitly to define our axioms for sets of elements (items) of universals and particulars of ontologies. The early paper by Noy and McGuiness [1], page 18, raises the question "An instance or a class?". We want to find an answer to this question by looking at the abstraction levels of ontologies and through modeling guidelines. The book by Arp,

[2] Please note: The terms "intension" and "intention" are related but have different meanings: while "intension" refers to the internal meaning or definition of a concept, "intention" refers to the purpose or goal behind a person's actions.

Smith and Spear on [2], page 145 ff, contains "Some Examples of Axioms". For example, they state that "Something is a universal if it is instantiated by something". On page 13 they explain that universals are "*repeatable*" while particulars are not. We think it would be worthwhile to provide an extended formal definition of what *instantiation* is. Also, the WonderWeb deliverable [4] by Masolo et al. contains many ontology definitions and an extensive axiom system. We use the basic formalisms of *Formal Concept Analysis* (FCA) described by Uta Priss [12], by Bernhard Ganter [13], and by Lübbert and Zeh [14] to specify axioms for *subsumption*. We think that none of the references discusses subsumption to the extent that we think it is necessary to capture order relations of universals.

Guarino in [9] introduces the notion of an *Identity Criterion* (IC), which allows to identify the identity or non-identity of two knowledge entities based on their properties and he states that "ICs for classes corresponding to natural language words are difficult or impossible to express". It seems important to us to investigate how the IC criterion can be extended to classes in general.

Challenges and limitations: In comparable approaches, the terminology is very inconsistent. In addition, the common methods for assigning names to ontology elements generally do not allow the type of ontology element to be identified. This is often the cause of misinterpretations of the modeling. The challenge is therefore to find a method for naming ontology elements that is short and concise and at the same time enables disambiguation. Furthermore, comparable axiom systems implicitly assume an understanding of basic ontology elements without providing a mathematically sound axiom system. For example, to the best of the author's knowledge, there are no axioms for the definition of the intension and extension of classes and their relationship.

3 Foundations

We define a *Knowledge Graph* (KG) as a three-dimensional data structure which is spanned by the dimensions *Generalization*, *Aggregation* and *Particularization* shown in Figure 1. Each KG represents an ontology $\boldsymbol{O}$. In the example, the set of classes is $\mathcal{C}$ = {C1, ..., C6}, the set of attributes[3] is $\mathcal{A}$ = {A1, ..., A6}, the set of binary relationship types[4]

[3] We use "attribute" and "data property" as synonyms.

[4] We also use "binary relationship type" and "object property" as synonyms. Note that, in contrast to our usage, in Object Oriented Programming (OOP) properties are not first-class entities.

is $\mathcal{R}$ = {(C4, ◊OP, C2)} and the set of particulars is $\mathcal{E}$ = {P1, ..., P7). The class hierarchy ($\mathcal{C}$, $\leq_\wedge$), the relationship type hierarchy ($\mathcal{R}$, $\leq_\Diamond$), and the attribute type hierarchy ($\mathcal{A}$, $\leq_.$) are organized along the *Generalization* dimension. Each class aggregates data property and object property definitions along the *Aggregation* dimension. In the particulars layer the attribute-value assignments to entities and the connections of entities to others are also organized along the *Aggregation* dimension. The generalization-aggregation plane constitutes the Schema Layer (SL). The particulars p ∈ $\mathcal{E}$ which are lying in the *Particulars Layer* (M_0 = PL) are connected to their classes in the *Schema Layer* (M_1 = SL) with, e.g., `(P7, »pof, C6)`. Thus, particulars are instantiated along the *Particularization* dimension into the particulars layer plane.

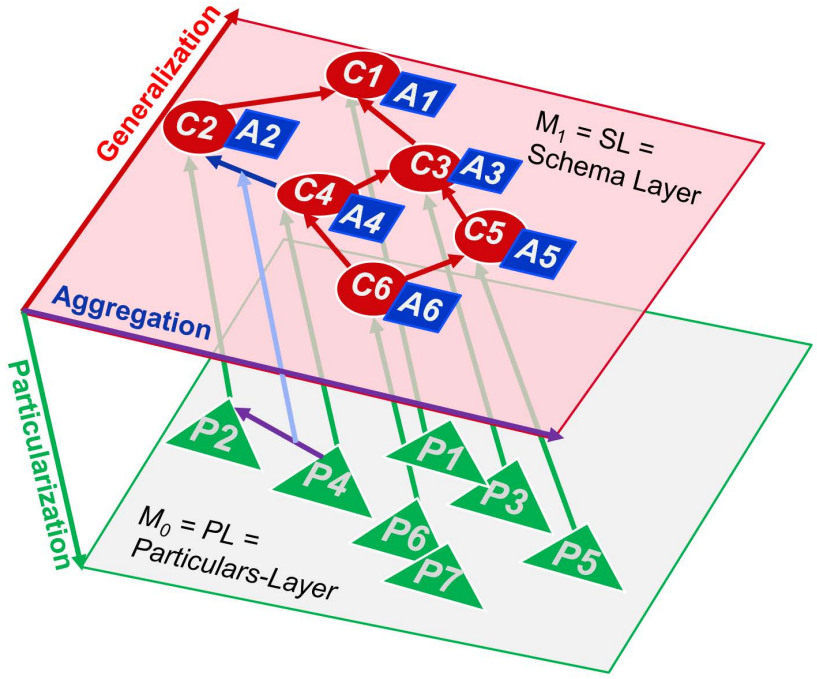

Fig. 1 Ontology Layers

Data Properties (DP) represent *intrinsic* properties of concepts and have a an *Atomic Data Type* adt ∈ $\mathcal{ADT}$ = {:String, :Integer, :Decimal, :Float, :Boolean, :Date, :Time, :anyURI, ...}. A class C aggregates declarations for data property A in the form of the *Data Property Definition* (DPD) `dpd = (C, A, adt)` along the *Aggregation* dimension of the SL layer. In a

DPD the class `C` plays the role of the domain and the `adt` plays the role of the range.

Object Properties (OP) represent *extrinsic* properties of concepts. *Object Property Definitions* (OPD) between the domain class `C` and the range class `D` are also defined in the SL layer in the form `opd = (C, ◊op, D)`. *Object Property Instantiations* (OPI) can take place between particulars `P` and `Q` in the PL Layer, or between a particular P and a class C between the PL and the SL layer. The former takes the form `(P, »op, Q)` and the latter takes the form `(P, »op, C)` or `(C, »op`$^{-1}$`, P)`. The blue arrow from `C4` to `C2` in figure 1 indicates an `opd = (C4, ◊op, C2)` while the purple arrow indicates an `opi = (P4, »op, P2)`. The light blue arrow between the opi and the opd indicates, that the opi is an instantiation of the opd which we note with `(opi, »opi, opd)`. This state of affairs can be expressed with `((P4, »op, P2), »opi, (C4, ◊op, C2))`. As with the particularization of particulars of classes, the instantiation of object properties takes place between the SL and the PL layers.

Before answering any of the questions raised in the Introduction section 1, we need to introduce some naming conventions and graphical notations. These are mainly based on the author's article [15].

Name Sets: Let N be the set of strings containing all possible sequences of arbitrary length, including the empty sequence, that can be generated from the characters of a given character set. As our key naming convention, the name of an ontological entity always uses the index of the designating set as its prefix: $N_x = \{xn \mid n \in N\}$. To unambiguously denote the names of ontological concepts we introduce the following name sets: $N_{\hat{}}$ is the set of class names, $N_{.}$ is the set of data property names, $N_{\Diamond}$ is the set of object property Names, $N_{>}$ is the set of particular names, $N_{»}$ is the set of relator names, $N_{\circ}$ is the set of process names, $N_{\tilde{}}$ is the set of function names, and the union of the sets is $N^* = \cup \{ N_x \mid x \in \{ \hat{}, ., \Diamond, >, », \circ, \tilde{} \}\}$. In short, the symbols are motivated to evoke associations with hierarchies, properties, processes, things in flux, and references / pointers. This means that the name sets defined in this way form the vocabulary of a meta-language for compact and expressive notations. For example, the RDFS definition of a class `^Car` based on our naming conventions is:

```
^Car rdf:type rdfs:Class.
```

Knowledge Graphs: We represent an ontology by a *Knowledge Graph* (KG). A Knowledge Graph is defined as KG ⊆ N x N x N. As in RDF[5] each

[5] https://www.w3.org/TR/rdf11-concepts/#section-triples

element is a triple of the form `(s, p, o)` which we call *Atomic Knowledge Expression* (AKE). First, we define the *inverse axiom* which we think is very important for a *General Ontology Theory* because it concerns every triple of a KG. For every property p there exists a property $q = p^{-1}$ such that the following axiom holds:

Definition 1 (InvAx = Inverse Axiom).
(s, p, o) ∈ KG ⟺ (o, q, s) ∈ KG.

Filters: Let M be a set and ≤ be a non-strict, acyclic, transitive ordering relation. The pair (M, ≤) is a non-strictly ordered set. A *principal filter* and a *principal ideal* are special subsets of a partially ordered set (see Ganter and Wille, [16]). We use principal filters to model hierarchies of super-elements symbolized by the ∵ icon, and to model hierarchies of sub-elements with principal ideals, symbolized by the ∴ icon:

Definition 2 (principal filter).
∵τa = {x ∈ M/ a ≤τ x} is the principal filter of τa.

Definition 3 (principal ideal).
∴τa = {z ∈ M/ z ≤τ a} is the principal ideal of τa.

Hierarchies: As a foundation for ontological engineering, we provide the minimal ontology blueprint called *O4Top* as shown in Figure 2 as an implementable artifact in the sense of [8]. We do not consider O4Top as a top-level ontology, but as a blueprint according to which ontologies should be constructed. For the visualization of knowledge graphs called *OntoGraphs* we use the symbols and naming conventions which have been introduced by Bense in [15]. The different colors and shapes for the various ontology elements enable modelers to understand the models more quickly and help him to identify and correct errors, and to reduce developing costs. Compared to other top-level ontologies, the O4Top ontology blueprint contains as a minimal set of ontological elements `^Class`, ◊`ObjectProperty`, `.DataProperty`, and »`^Resource`. Having »^Resource as the most generic modeling entity was motivated by Allemang et al. who state in [3], page 37: "a resource can be anything that someone might want to talk about". *Classes* correspond to formal concepts in Formal Concept Analysis as we will discuss in detail below. *Object Properties* (OP) represent binary relationship types between classes. A *particular* is the instantiation (see section Instantiation 3.2) of a class and bound to the class with the binary relationship type »pof (particularOf). A *relator* is the instantiation of a binary relationship type or, in the case of n-ary relations, the binding of one particular to n other particulars.

For completeness, we mention the »^UResource and the additional types of data properties ..MetadataProperty, .ΔTransparentDataProperty, and .^UniversalsDataProperty for streamling conceptual modeling and Meta-Level Modeling (MLM), which are explained in the author's article [17] and are not relevant to the discussion in this paper.

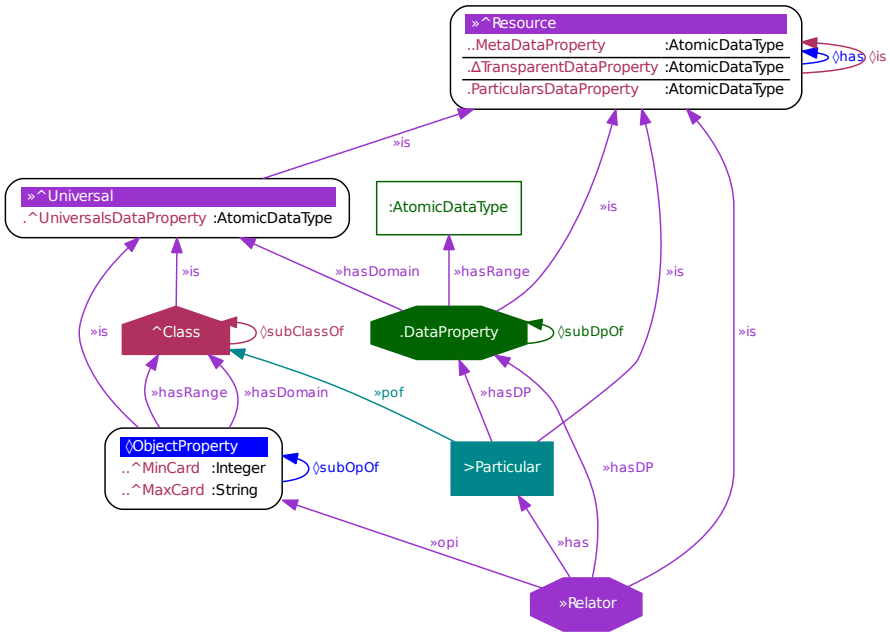

Fig. 2 O4Top Ontology Blueprint

For an ontology $\mathcal{O}$, with $\mathcal{C}$ we denote its set of classes, with $\mathcal{A}$ its set of *Data Properties* (DP), with $\mathcal{R}$ its set of *Object Properties* (OP), with $\mathcal{E}$ (*Extension*) its set of class particulars, and with $\mathcal{EE} = \mathcal{E}$ X $\mathcal{E}$ its set of object property instances. The OPs ◊subClassOf, ◊subDpOf and ◊subOpOf are acyclic, transitive relationship types and form hierarchies using the order relations $\leq_{\Diamond}$, $\leq_{\wedge}$, and, $\leq_{.}$:

- x $\leq_{\wedge}$ y $\Leftrightarrow$ $\exists$x, y $\in \mathcal{C}$: (x, »subClassOf, y) $\in$ KG $\vee$ (y, »hasSubClass, x) $\in$ KG
- x $\leq_{\Diamond}$ y $\Leftrightarrow$ $\exists$x, y $\in \mathcal{R}$: (x, »subOpOf, y) $\in$ KG $\vee$ (y, »hasOp, x) $\in$ KG
- x $\leq_{.}$ y $\Leftrightarrow$ $\exists$x, y $\in \mathcal{A}$: (x, »subDpOf, y) $\in$ KG $\vee$ (y, »hasDp, x) $\in$ KG

- The pair $\mathcal{C}\therefore = (\mathcal{C}, \leq_{\wedge})$ is the class hierarchy structure where for each c $\in \mathcal{C}$ the subclass hierarchy of c is defined as $\therefore$c. Its top element is sup($\therefore$c) = c and sup($\mathcal{C}$) = ^Class. The set of super classes of c is defined as $\because$c.

– The pair $\mathcal{R}\therefore = (\mathcal{R}, \leq_{\Diamond})$ is the hierarchy structure of object properties where for each r $\in \mathcal{R}$ the sub OP hierarchy of r is defined as $\therefore$r. Its top element is sup($\therefore$r) = r and sup($\mathcal{R}$) = ◊ObjectProperty. The set of super OPs of r is defined as $\because$r.
– For data properties the pair $\mathcal{A}\therefore = (\mathcal{A}, \leq_{\cdot})$ is the hierarchy structure of data properties where for each a $\in \mathcal{A}$ the sub DP hierarchy of a is defined as $\therefore$a. Its top element is sup($\therefore$a) = a and sup($\mathcal{A}$) = .DataProperty. The set of super DPs of a is defined as $\because$a. E.g., .GivenName $\leq_{\cdot}$ .PersonName $\leq_{\cdot}$ .Name

3.1 Properties

We distinguish between Data Properties (DP) and Object Properties (OP) and their *Data Property Definitions* (DPD) and *Object Property Definitions* (OPD). For both we give a short and a full definition pattern. The full two-triple definition patterns using the OPs ◊`hasDomain` and ◊`hasRange` are equivalent to the short one-triple definition patterns.

Data Property Definitions (DPD): Intrinsic attributes of entities are treated as data properties (DP) according to the Semantic Web / OWL / RDF convention. These include data properties such as `color`, `length`, `weight`, `age`, etc., which define the corresponding quality dimensions. The full definition of a DP is built with two triples, e.g., `(.LicensePlateNbr, »hasDomain, ^Vehicle)` and `(.LicensePlateNbr, »hasRange, :String)`. The short form of a DPD determines the associated type for the class, e.g. `(^Vehicle, .LicensePlateNbr, :String)` which instantiates, for example, to `(>Toms_Lambo, .LicensePlateNbr, DO-IT 1954)`. Let a $\in$ `N`$_{\cdot}$ be the name of a data property, `adt` $\in \mathcal{ADT}$ an Atomic Data Type, and u $\in$ `N` be the name of a universal. Both templates of the DPDs are equivalent according to the following rule:

Ax 01 (Data Property Definition Axiom)**.** *(u, a, :adt)* $\in$ *KG* $\Leftrightarrow$ *(a, »hasDomain, u)* $\in$ *KG* $\wedge$ *(a, »hasRange, :adt)* $\in$ *KG*

Object Property Definitions (OPD): Extrinsic properties of entities have been introduced as *Object Properties* (OP) according to the convention in the Semantic Web / OWL / RDF. Object properties bind entities to other entities or to themselves. The full definition of an object property, e.g., ◊`hasMotor`, is defined by the two triples `(`◊`hasMotor, »hasDomain, ^Car)` and `(`◊`hasMotor, »hasRange, ^Motor)`. The short form of an *Object Property Definition* is `(^Car,` ◊`hasMotor, ^Motor)`. As an example from a vehicle ontology we have `(>Toms_Lambo, »hasCombMotor, >MTR_TL)` where

◊hasCombMotor is a subobject property of ◊hasMotor. Let ◊op ∈ $N_◊$ be the name of an object property, and c, d ∈ $N_^$ be the names of classes. Both templates of OPDs are equivalent by the rule:

Ax 02 (Object Property Definition Axiom)**.** *(c, ◊op, d) ∈ KG ⟺ (◊op, »hasDomain, c) ∈ KG ∧ (◊op, »hasRange, d) ∈ KG*

3.2 Instantiation

The smallest instantiation step is to create or update a (s, p, o) triple. There either a data property or an object property is involved. The first requires a *Data Property Definition* (DPD) as template, the second an *Object Property Definition* (OPD). Instantiation patterns specify what form the resulting triple must have. Now we define patterns for *Data Property Instantiatio*n (DPI) and *Object Property Instantiation* (OPI) where a ∈ $\mathcal{A}$ is a Data Property, adt ∈ $\mathcal{ADT}$ is an Atomic Data Type, u is a universal either from the set $\mathcal{C}$ of classes or from the set $\mathcal{R}$ of object properties, ^C and ^D are classes, and v is a value from the value set of attribute a:

Data Property Instantiation (DPI): Then each DPI must map to one of the following patterns where the symbol ⊨ relates the property definition template to its allowed property instantiations. The first rule expresses that a DPD can be used to instantiate particulars or universals. The second rule expresses that binary and n-ary relations[6] between objects can also have data properties. We refer to the next section where we will define the function pof() to return the set of particulars of a class .

- (u, A, adt) ⊨ (x, A, v) where (x, »pof, u) or x ∈ ∴u
- (◊OP, A, adt) ⊨ (»OPI, A, v) for relators with (»OPI, »opi, ◊OP)

Object Property Instantiation (OPI): An *Object Property Definition* (OPD) has the short form (^C, ◊OP, ^D). The classes ^C and ^D may be identical. Then the possible *Object Property Instances* (OPI) for *Particular-Particular Relationships* (PPR) and for *Class-Particular Relationships* (CPR) are:

- OPI: Let >P1 ∈ pof(^C) and >P2 ∈ pof(^D), then (>P1, »OP, >P2) is an object property instance of (^C, ◊OP, ^D).
- PPR: (>P, »isOPOf, ^C), or also its inverse (^C, »hasOP, >P) both mean that for an OPD (^Person, ◊isDesignerOf, ^Car) the particular >P on the particulars layer is connected with ^C on the schema layer, for example, (>Ferdi_Porsche, »isDesignerOf, ^VW_Beetle) and (^VW_Beetle, »hasDesigner, >Ferdi_Porsche).

[6] RDFS and OWL do not allow n-ary relations.

Object Property Instantiation Rules (OPIR): Be $\Diamond_{\texttt{OPOf}} = \Diamond_{\texttt{OP}}^{-1}$, and $\texttt{>P1} \in \texttt{pof(\^{}C)}$, and $\texttt{>P, >P2} \in \texttt{pof(\^{}D)}$. For PPRs and for CPRs and their inverses we get as OP instantiation rules:

- PPR: $\texttt{(\^{}C,}\Diamond_{\texttt{OP}}\texttt{,\^{}D)} \models \texttt{(>P1,»OP,>P2)}$; $\texttt{(\^{}D,}\Diamond_{\texttt{OPOf}}\texttt{,\^{}C)} \models \texttt{(>P2,»OPOf,>P1)}$
- CPR: $\texttt{(\^{}C,}\Diamond_{\texttt{OP}}\texttt{,\^{}D)} \models \texttt{(\^{}C,»OP,>P)}$; $\texttt{(\^{}D,}\Diamond_{\texttt{OPOf}}\texttt{,\^{}C)} \models \texttt{(>P,»OPOf,\^{}C)}$

4 Ontological Sets and their Axioms

With the help of figure 1 we will discuss in the following which laws result from the ontology structures. These laws can then be used to check the structural correctness of ontologies. Let P(M) be the powerset of a set M, C a class, T $\subseteq \mathcal{C}$ a subset of classes, and $\therefore$C the set of classes of the class hierarchy of C. To compute the set of particulars of a class we define the function pof: $\mathcal{C} \rightarrow$ P($\mathcal{E}$). To compute the set of particulars of a set of classes we define the function sop: P($\mathcal{C}$) $\rightarrow$ P($\mathcal{E}$) with:

Definition 4 (Particulars OF a class).
pof(C) := { y | (y, »pof, C) ∈ KG}.

Definition 5 (Set Of Particulars of classes).
sop[T] := ∪ { pof(C) | C ∈ T}.

Then, we have as examples with $\therefore$C3 = {C3, C4, C5, C6} $\subseteq \mathcal{C}$ as the set of particulars of a single class pof(C3) = {P3}, pof(C4) = {P4}, pof(C5) = {P5}, and pof(C6) = {P6, P7}. It follows that the SOPs of a class hierarchy of C3 is sop[$\therefore$C3] = sop[{C3, C4, C5, C6}] = {P3} $\cup$ {P4} $\cup$ {P5} $\cup$ {P6, P7} = {P3, P4, P5, P6, P7}.

Sets of Elements of Classes (SOE): The set of particulars of a class differs from the *Set Of Elements* (SOE) of a class in that particulars can be *fully instantiated* (s. definition 8 below) in the class. For each particular, however, there exists at least one class C $\in \mathcal{C}$ in which it can be fully instantiated. Then the function soe: $\mathcal{C} \rightarrow$ P($\mathcal{E}$), which computes the set of elements of class C, can be defined as:

Definition 6 (Set Of Elements).
soe(C) = ∪ {pof(X) | X ∈ ∴C}.

We denote the extension (ext) of a class C by extA(C) := $\underline{\text{C}}$:= soe(C). This leads to the axiom that a set of particulars of the class hierarchy of C is equivalent to the set of elements of the class C:

Ax 03 (Particulars-Elements-Correspondence)**.** *soe(C) = sop[∴C].*

Then we have for single classes sop(C4) = {P4} and sop(C6) = {P6, P7} and for their extensions $\underline{C4}$ = soe(C4) = sop(∴C4) = {P4, P6, P7}, and $\underline{C6}$ = soe(C6) = sop(∴C6) = {P6, P7}, and so on. Note, e.g., that P3 is an element of C1 but not a particular of C1. In general, all particulars of sublass C of a class D with D ≠ C and C $\leq_{\wedge}$ D are elements of D but not particulars of D.

Set of Attributes of Classes (SOA): Let ∵$_c$ be the set of superclasses of c including c, and $adt \in \mathcal{ADT}$ an atomic data type, and $a \in \mathcal{A}$ a data property. To compute the set of Data Property Definitions (DPD) of a class and its superclass hierarchy and associated set of attributes we define the functions sad: $\boldsymbol{C} \rightarrow P(\boldsymbol{A})$ and soa: $\boldsymbol{C} \rightarrow P(\boldsymbol{A})$ as follows:

Definition 7 (SAD = Set of Attribute Definitions).
sad(C) := { a | (C, a, adt) ∈ KG ∨ (a, »hasDomain, C) ∈ KG}.

Then, the function soa() for computing the set of attributes in the hierarchy of superclasses of a class C is defined as:

Definition 8 (SOA = Set Of Class Attributes).
soa(C) := ∪ {sad(X) | X ∈ ∵C}.

The *Set Of Attributes* (SOA) of a superclass hierarchy of C is also denoted as the *intension* of the class C with intA(C) := soa(C). It is important to realize that the instantiation of objects with data properties does not have to take on the full scope that would be possible based on the definition in the ontology schema. Therefore, we introduce the following definitions: Let x be a particular of the class C with (x, »pof, C) ∈ KG. If all attributes a ∈ intA(C) of an entity x are instantiated (have been assigned values), then we say "the entity x is *fully instantiated*", otherwise, "the entity x is *fully instantiable* with attributes from intA(C)".

So, the intension of a class is the set of attributes with which a particular of the class can be fully instantiated. That is, soa(C) is the set of attributes that have a Data Property Definition (DPD) in C or in the superclasses of C. Then we have, for example, sad(C1) = {A1}, sad(C2) = {A2}, sad(C3) = {A3}, ..., sad(C6) = {A6}. Furthermore, we have soa(C1) = {A1} ⊆ soa(C2) = {A1, A2}, soa(C3) = {A1, A3}, and soa(C3) ⊆ soa(C4) = {A1, A3, A4}, soa(C5), and soa(C4) ⊆ soa(C6) = {A1, A3, A4, A5, A6}. From this we already see that the smaller the intension of a class intA(C), the larger is its extension extA(C) = $\underline{C}$. Then, e.g., a particular P2 of C2 is fully instantiable with the set of attributes soa(C2) = {A1, A2}. This is equivalent to (P2, »pof, C2) ∈ KG. However, neither the attribute A1 nor A2 need to have a value assigned to P2.

In the following, in analogy to data properties we give set definitions for object properties. The set of outgoing object properties of a class C is defined as the set of OPs where C is the domain OP. This set is also denoted as the OP intension of the class C with intR(C) := sdp(C). We define sdp: $\boldsymbol{C} \rightarrow P(\boldsymbol{R})$ as:

Definition 9 (SDP = Set of Domain OPs).
sdp(C) := { ◊op | (C, ◊op, D) ∈ KG ∨ (◊op, »hasDomain, C) ∈ KG}.

In analogy to a class hierarchy the hierarchy of Object Properties (OP) is defined as ($\boldsymbol{R}$, $\leq_{\Diamond}$) and has the set of OPs ∴R := {r ∈ $\boldsymbol{R}$ | r $\leq_{\Diamond}$ R} with x $\leq_{\Diamond}$ y ⇔ x, y ∈ $\boldsymbol{R}$: (x, »subOpOf, y) ∈ KG. Then, for the set of object property particulars $\mathcal{EE}$ = $\mathcal{E}$ X $\mathcal{E}$ of an OP ◊R we define a function soopp: $\mathcal{R} \rightarrow$ P($\mathcal{EE}$) with:

Definition 10 (Set Of OP Particulars).
soopp(◊R) := {(x, y) | (x, »R , y) ∈ KG }.

To compute the set of elements of an OP hierarchy, we define the function soope: $\mathcal{R} \rightarrow$ P($\mathcal{EE}$). We also denote $\underline{\text{R}}$[7] := extR(R) := soope(R) as the extension of R.

Definition 11 (Set Of Object Property Elements).
soope(R) := ∪ {soopp(r) | r ∈ ∴R}.

The set of pairs (x, y) of arbitrary OPs where C is the domain class is denoted as the OP extension of C with extR(C):= soopi(C). With soopi: $\boldsymbol{C} \rightarrow P(\boldsymbol{EE})$ we define:

Definition 12 (Set Of Object Property Instances).
soopi(C) := {(x, y) | (x, »r, y) ∈ KG ∧ x ∈ pof(C) ∧ ◊r ∈ sdp(C)}.

Figure 3 shows a small example from the domain of cars and motors. To the left of the schema layer we see a subclass hierarchy of cars which is based on their type of motorization, to the right the corresponding subclass hierarchy of motors. Both subhierarchies contain with HybridCar and HybridMotor a class with multiple inheritance. The corresponding pairs of classes are connected by the OP definitions (Car, ◊hasMotor, Motor), (CombCar, ◊hasCombMotor, CombMotor), (ElectricCar, ◊hasElectricMotor, ElectricMotor), and also (HybridCar, ◊hasHybridMotor, HybridMotor). Each class has a particular which is connected by »pof to its class. For the class extensions we have:

[7] Note that the underline is essential for the definition of an extension.

– HybridCar = {MP}, CombCar = {TL, MP}, ElectricCar = {MT, MP}, Car = {HP, TL, MT, MP},
– HybridMotor = {MMP}, CombMotor = {MTL, MMP}, ElectricMotor = {MTT, MMP}, Motor = {MHP, MTL, MTT, MMP}.

The set of Object Property Instances (OPI) is formed by the triples (HP, »hasMotor, MHP), (TL, »hasCombMotor, MTL), (MT, »hasElectricMotor, MTT), and (MP, »hasHybridMotor, MMP). Then, soopp (◊hasMotor) = { (HP, MHP) }, soopp (◊hasCombMotor) = { (TL, MTL) }, soopp (◊hasElectricMotor) = { (MT, MTT) }, and soopp (◊hasHybridMotor) = { (MP, MMP) }. From this we can derive the extensions of the OPs as
– ◊hasHybridMotor = {(MP, MMP)}
– ◊hasElectricMotor = {(MT, MTT), (MP, MMP)}
– ◊hasCombMotor = {(TL, MTL), (MP, MMP)}
– ◊hasMotor = {(MP, MMP), (MT, MTT), (TL, MTL), (MP, MMP)}

Then, for the hierarchy and the extensions of the OPs we have
– ◊hasHybridMotor $\leq_{\Diamond}$ ◊hasElectricMotor,
– ◊hasHybridMotor $\leq_{\Diamond}$ ◊hasCombMotor,
– ◊hasCombMotor $\leq_{\Diamond}$ ◊hasMotor,
– ◊hasElectricMotor $\leq_{\Diamond}$ ◊hasMotor
– ◊hasHybridMotor ⊆ ◊hasElectricMotor,
– ◊hasHybridMotor ⊆ ◊hasCombMotor,
– ◊hasCombMotor ⊆ ◊hasMotor,
– ◊hasElectricMotor ⊆ ◊hasMotor

For example, for electric cars we have (ElectricCar, ◊hasElectricMotor, ElectricMotor) ∈ KG ∧ (MT, »hasElectricMotor, MTT) ∈ KG → MT ∈ pof(ElectricCar) ∧ MTT ∈ pof(ElectricMotor) ⇔ (MT, »pof, ElectricCar) ∈ KG ∧ (MTT, »pof, ElectricMotor) ∈ KG or in general with (C, ◊R, D) ∈ KG ∧ (P, »R, Q) ∈ KG → P ∈ pof(C) ∧ Q ∈ pof(D) ⇔ (P, »pof, C) ∈ KG ∧ (Q, »pof, D) ∈ KG.

4.1 Class Subsumption Axiom

From the above examples we see soe(C4) ⊆ soe(C3) ⇔ (C4, $\leq_{\wedge}$, C3) ⇔ int(C3) = soa(C3) ⊆ int(C4) = soa(C4). This is consistent with the axioms in *Formal Concept Analysis* (FCA) as described in [12], because the set $\mathcal{E}$ corresponds to *formal objects* (Gegenstände) and $\mathcal{A}$ corresponds to *formal attributes* (Merkmale). For I ⊆ $\mathcal{E}$ x $\mathcal{A}$, the relation $\leq_{\wedge}$ is called *formal conceptual ordering* on the set of all formal concepts $\mathcal{B}(\mathcal{E}, \mathcal{A}, I)$. So, in general we can define an extended *class subsumption axiom* (CSAx) as:

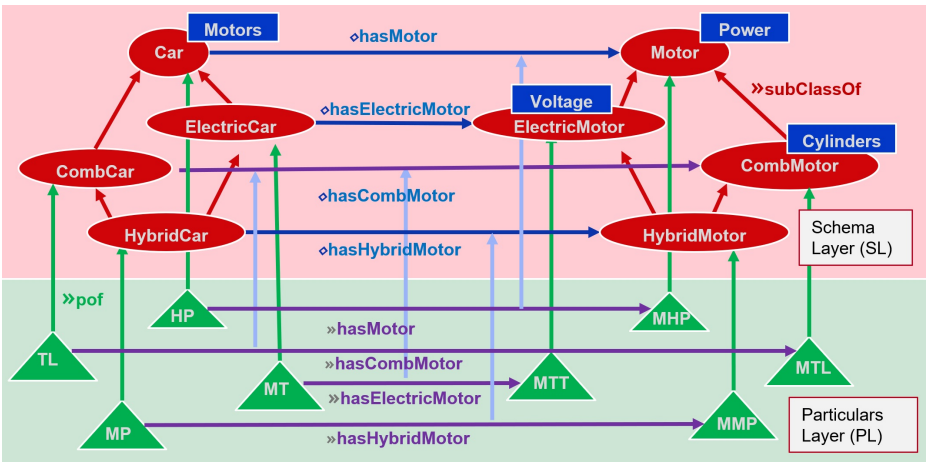

Fig. 3 Object Property Hierarchy for Cars and Motors

Ax 04 (CSAx = Class Subsumption Axiom)**.** $soe(C) \subseteq soe(D) \iff C \leq_\wedge D \iff sop[\therefore C] \subseteq sop[\therefore D] \iff soa(D) \subseteq soa(C)$.

As pointed out in [14], we can consider an ontology to be correct with respect to the structure of its classes, attributes, and particulars, if elements from $\mathcal{C}$, $\mathcal{A}$, and $\mathcal{E}$ constitute a *formal concept*. As an important result we find that only from the set of attributes $\mathcal{A}$ of particulars $\mathcal{E}$ we can infer its class hierarchy structure $\mathcal{C}\therefore = (\mathcal{C}, \leq_\wedge)$. Therefore, it is possible with FCA methods to automatically identify class candidates based on simple cross-tables, where it is only necessary to mark with an X those objects e ∈ $\mathcal{E}$ which have attributes a ∈ $\mathcal{A}$.

The complete class hierarchy $\mathcal{C}$ of an ontology $\mathcal{O}$ can usually consist of several sub-hierarchies $\therefore C_i \subseteq \mathcal{C}$ which model different areas of the application domain, e.g., $\therefore$C1 = $\therefore$Person, $\therefore$C2 = $\therefore$Car, $\therefore$C3 = $\therefore$Motor etc. An attribute A can only be defined once in $\mathcal{C}$. The reason for this is that otherwise the subsumption axiom Ax 04 would be violated. I.e., if the attribute A occurs more than once in $C_x \in \mathcal{C}$ and $C_y \in \mathcal{C}$, then it must be moved to the smallest / lowest common superclass C_z of C_x and C_y, so that both $C_z \in \because C_x$ and $C_z \in \because C_y$ hold.

4.2 Relationship Type Subsumption Axiom

According to Lübbert and Zeh [14], page 8 ff, the subsumption principle can now also be used to automatically detect hierarchies of Relationship Types (RT). We define the subsumption axiom for object properties based on the insight that an object property ◊R is a direct or indirect subobject

property of the object property ◊S if its elements are a subset of the elements of its superobject property:

Ax 05 (RTSAx = Relationship Type Subsumption Axiom)**.** *soope(◊R) ⊆ soope(◊S) ⇔ $\underline{◊R} \subseteq \underline{◊S}$ ⇔ ◊R ≤◊ ◊S.*

Now let us consider the consequences that follow from OP ordering relations. Let ◊T be an object property and ◊R, ◊S ∈ ∴◊T with ◊R ≤◊ ◊S and ◊R ≠ ◊S, and C, D ∈ $\mathcal{C}$, and C1, C2 ∈ ∴C, D1, D2 ∈ ∴D, (C1, ◊R, D1) ∈ KG, and (C2, ◊S, D2) ∈ KG. Then, with p ∈ pof(C1), q ∈ pof(D1), and (p, q) ∈ $\underline{◊R}$ follows (p, q) ∈ $\underline{◊S}$. This also means with soe(C2) = $\underline{C2}$ and soe(D2) = $\underline{D2}$ that p ∈ $\underline{C2}$ ∧ q ∈ $\underline{D2}$ and this is exactly the case if C1 ≤∧ C2 ∧ D1 ≤∧ D2 where also holds $\underline{C1} \subseteq \underline{C2}$ ∧ $\underline{D1} \subseteq \underline{D2}$. This is also true for the cases where C1 = C2 and D1 = D2. This means that pairs of particulars that make up the extension of an OP automatically force the arrangement in the respective domain and range class of the OP. For example, for the OPs hasMotor, hasElectricMotor, hasCombMotor and hasHybridMotor each have a specific motor type as the range entity of the pair (x, y). Since the motor types are arranged in an inheritance hierarchy ∴Motor, the domain entities are also automatically arranged in an isomorphic inheritance hierarchy. This leads to the following consideration: Even if the class Car has no subclass, the ordering relation for the subclass hierarchy ∴Car can be inferred from the OP hierarchy of ◊hasMotor, e.g., in (x, »hasElectricMotor, y) the object y must be a particular of ElectricMotor, and from this it can be inferred that x must be an ElectricCar. This leads to the definition of *properly suspended relationship types*:

Definition 13 (Properly Suspended RT).
Be ◊R ≤◊ ◊S. ◊R is properly suspended under ◊S if: ((C1 ≤∧ C2) ∧ (D1 ≤∧ D2)) ⇔ (($\underline{C1} \subseteq \underline{C2}$) ∧ ($\underline{D1} \subseteq \underline{D2}$)) .

Then we claim a subsumption theorem for relationship hierarchies in analogy to axiom 03. The statement is: For a pair of object properties where one OP is a subobject property of the other (◊R ≤◊ ◊S), the domain and range classes involved must be superclasses and subclasses of each other, respectively.

Theorem 1 (RTSTh = RT Subsumption Theorem).
◊R ≤◊ ◊S ⇒ (C1 ≤∧ C2) ∧ (D1 ≤∧ D2).

Proof of RTSTh: Suppose that C1 ≤∧ C2 or D1 ≤∧ D2 does not hold. Then we would have C1 >∧ C2 or D1 >∧ D2, and $\underline{C1} \supset \underline{C2}$ or $\underline{D1} \supset \underline{D2}$.

This would imply that the more general class has fewer elements than one of its subclasses, e.g., there are more ElectricMotors than Motors or more ElectricCars than Cars. This leads to a contradiction. □

Applying these principles, we can now use Formal Concept Analysis (FCA) to extend the lattices of classes to lattices of their object properties. As formal objects we use the set of pairs $\mathcal{EE}$= {(x, y) | x, y ∈ ***E***} and as formal attributes the set $\mathcal{R}$. Then, e.g., from object property instances such as (x, »hasCombMotor, y), we find order relations of the form »hasHybridMotor $\leq_\Diamond$ »hasCombMotor, »hasElectricMotor $\leq_\Diamond$ »hasMotor. Analogous to the class hierarchy, I ⊆ $\mathcal{EE}$ x $\mathcal{R}$ and the relation $\leq_\Diamond$ is the *formal concept ordering* and the set of all formal relationship type concepts is $\mathcal{BR}(\mathcal{EE}, \mathcal{R}, I)$.

In the example (^Car, ◊hasMotor, ^Motor), the class ^Car can be understood as an aggregator and ^Motor as an aggregatee. In the example (^Artist, ◊isCreatorOf, ^Artifact) the situation is reversed (see lattices of relationship types in [14], page 8 ff). Here ◊isCreatorOf can be understood as an OP that classifies ^Artist in relation to the type of the produced ^Artifact, e.g. with ◊isSculptorOf $\leq_\Diamond$ ◊isCreatorOf, ◊isOilPainterOf $\leq_\Diamond$ ◊isPainterOf $\leq_\Diamond$ ◊isCreatorOf, and ◊isCreatorOf = ◊hasCreator^{-1}. That is why in our interpretation, the general OPs ◊has and ◊is from figure 2 are considered dual / orthogonal to each other. All OPs that are derived from ◊has can be attributed to aggregation, all OPs that are derived from ◊is can be attributed to classification / generalization.

4.3 Class Identity Axiom

As an extended axiom, we define that a class C is identical to another class D if both their attribute and relationship type intensions, as well as their element and object property extensions are identical. Be C, D ∈ ***C***.

Ax 11 (CIdAx = Class Identity Axiom)**.** *(intA(C) = intA(D)) ∧ (intR(C) = intR(D)) ∧ (C = D) ∧ (extR(C) = extR(D)) → C = D.*

5 Results

As a fundamental artifact in the sense of [8], with O4Top we have developed a minimal top ontology. This is the basis for an axiom system that formally describes the mathematical relationships between the elements of O4Top. It also contains definitions for the intension and extension of classes and laws for inheritance and instantiation. The Class

Subsumption Axiom 04 and the Relationship Type Subsumption Axiom 05 were derived from this as new laws. The RT Subsumption Theorem 1 was then used to prove that strict inheritance relationships apply between classes involved in sub-relationships. This also made it possible to define a Class Identity Axiom 11, which establishes a connection between the intensions and extensions of classes and the relationship types involved in them. Compared to other larger top-level ontologies such as BFO, DOLCE, GFO and UFO, the top-level ontology O4Top presented in the Axioms section 4 is minimal. We believe that O4Top is general and robust enough to model not only other top-level ontologies but also domain ontologies. Therefore, we also assume that the axiom systems developed elsewhere can still be used. In our opinion, the axioms discussed in the previous sections could be added under other ontological axiom systems. The utility of our method for the streamlining of conceptual modeling has been discussed in the author‘s paper [17] (The goal of design-science is utility: [8], page 80). Our method also supports the detection and elimination of inconsistencies in conceptual modeling with the help of build-and-evaluate loops (The goal of behavioral-science is truth: [8], page 78, 80).

6 Discussion

Guarino in [9] discusses a “Minimal Ontology of Universals”. We would map his universals to the modeling elements of our O4Top ontology blueprint as follows: ^Class and ◊ObjectProperty (relationship type) are our top universals. Our understanding of “Property” is similar in that we consider the “Attribution” to represent Data Properties (DP) and the other properties are Object Properties (OP). We think that “Type”, “Category”, and “Role” can be represented by ^Class. Our top universal ◊ObjectProperty corresponds to ”Relation”.

Guarino suggests having two separate ontologies for particulars and universals, keeping lexical items out of the domain. In our approach, particulars and universals are part of the same ontology. They just reside in the two different layers PL and SL. The YAGO ontology needs quadruples to represent so-called *reification graphs* ([10], page 10 ff.). This has the advantage in contrast to RDFS and OWL, that n-ary relations can be modeled easily. However, when using triple stores, there is a mapping overhead due to the additional reification requirements. On the other hand, we see the Yago approach as a confirmation that a general ontology theory like ours can get by with a small set of model elements and axioms.

Future Work: An ontology structure like $\mathcal{O} = (\mathcal{C}\therefore, \mathcal{R}\therefore, \mathcal{A}\therefore, \mathcal{E}, \mathcal{EE}, \mathcal{ADT})$ needs to be completed in future work. It contains similar components to that what Herre et al. define in [11], page 13, as a signature Σ = *(Cat,OCat,P,RCat,R; Ind, Obj ,Att,Rol ,Rel ;=, :: , inh, roleof)* of an ontology vocabulary. The building blocks of Guarino's, Herre's and YAGO's structures are similar to those in our approach and could be complemented by our axiomatics. Also, we plan to extend our minimal top level schema by elements which allow to model processes and events.

7 Conclusion

In this article, we have proposed a *General Ontology Theory* (GOT). The foundation is the O4Top ontology blueprint which is based on a minimal set of ontological concepts. Using the transitive relationship types subClassOf, subOpOf and subDpOf and applying the principal ideal filter, we derived the hierarchy structures for classes, object and data properties. After elaborating the difference between elements of classes and particulars of classes, and the relationship between extension and intension we were able to define the particulars-elements correspondence axiom 03. Then, applying axioms from Formal Concept Analysis, we obtained the extended class subsumption axiom 04. This allows a hierarchy of formal concepts to be derived only from the set of objects and the sets of attributes of those objects. That is, a modeler can start with a table of objects and their attributes as input and get the underlying hierarchy of formal concepts as result. Applying the relationship type subsumption theorem 1 it becomes possible to infer hierarchy relationships of formal concepts solely from object property instantiations, and to check the structural correctness of ontologies with respect to the ordering relations of object properties.

References

1. Noy NF and McGuinness DL. Ontology Development 101: A Guide to Creating YourFirst Ontology. Stanford University, Stanford, CA, 94305 2000. Available from: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf
2. Arp R, Smith B, and Spear AD. Building Ontologies with Basic Formal Ontology. The MIT Press, London, England 2015. ISBN: 978-0-262-52781-1
3. Allemang D, Hendler J, and Gandon F. 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
4. Masolo C, Borgo S, Gangemi A, Guarino N, and Otramari A. Wonderweb Deliverable D18. Ontology Library (final). ISTC-CNR c/o ISIB-CNR 2003

5. Guizzardi G, Benevides AB, Fonseca CM, Porello D, Almeida JPA, and Sales TP. UFO: Unified Foundational Ontology. Applied Ontology 1-3 2021. Available from: https://www.researchgate.net/publication/355735118_UFO_Unified_Foundational_Ontology
6. Selway M, Stumptner M, Mayer W, Jordan A, Grossmann A, and Schrefl M. A Conceptual Framework for Large-scale Ecosystem Interoperability and Industrial Product Lifecycles. Data Knowledge Engineering 109 2017 :85–111. DOI: 10.1016/j.datak.2017.03.006. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0169023X1730109X?via%3Dihub
7. Nunamaker JF, Chen M, and Purdin TDM. Systems Development in Information Systems Research. Journal of Management Information Systems, M.E. Sharpe Inc., Vol. 7, No. 3 1991 :89–106
8. Hevner AR, March ST, Park J, and Ram S. Design Science in Information Systems Research. MIS Quarterly Vol. 28 No. 1 2004 :75–105. Available from: https://www.researchgate.net/publication/201168946_Design_Science_in_Information_Systems_Research
9. Guarino N. Some Ontological Principles for Designing Upper Level Lexical Resources. First Int. Conference on Language Resources and Evaluation, Granada, Spain, 28-30 May 1998
10. Suchanek FM, Kasneci G, and Weikum G. Yago: A Large Ontology from Wikipedia and WordNet. 2007. Available from: https://pure.mpg.de/rest/items/item_1819068/component/file_1840695/content
11. Herre H, Heller B, Burek P, Hoehndorf R, Loebe F, and Michalek H. General Formal Ontology (GFO). 2006. Available from: https://www.onto-med.de/sites/www.onto-med.de/files/files/uploads/Publications/2006/herre-h-2006-a.pdf
12. Priss U. The Formalization of WordNet by Methods of Relational Concept Analysis. WordNet: An Electronic Lexical Database and Some of its Applications, MIT press 1998 :179–96. Available from: https://www.upriss.org.uk/papers/mitpaper.pdf
13. Ganter B. Begriffe und Implikationen. In: Gerd Stumme, Rudolf Wille (edt.), Begriffliche Wissensverarbeitung: Methoden und Anwendungen, Springer 2000. ISBN: 3-540-66391-6:1–24
14. Lübbert C and Zeh T. Skizze eines Verfahrens zur Erstellung von Ontologien mittels Formaler Begriffsanalyse. Informatik Spektrum (2022) 45 2021 :3–12. Available from: https://link.springer.com/content/pdf/10.1007/s00287-021-01397-1.pdf
15. Bense H. The Unique Predication of Knowledge Elements and their Visualization and Factorization in Ontology Engineering. Kutz O, Garbacz P (eds.), Proceedings of the Eighth International Conference (FOIS 2014), Rio de Janeiro, Brazil, Sept. 22-25 2014 :241–50. DOI: 10.3233/978-1-61499-438-1-251. Available from: https://ebooks.iospress.nl/publication/37972
16. Ganter B and Wille R. Formal Concept Analysis - Mathematical Foundations. 2nd Edition, Springer Berlin Heidelberg 2024. ISBN: 978-3-031-63421-5
17. Bense HJ. Streamlining Conceptual Modeling. Kohei Arai (edt.), Proceedings of the 2023 Intelligent Systems Conference (IntelliSys), Amsterdam, The Netherlands, Springer Nature, Lecture Notes in Networks and Systems 822 2023. ISBN: 978-3-031-47721-8:326–44. Available from: https://link.springer.com/chapter/10.1007/978-3-031-47721-8_22