Guideline on Developing Good Ontologies in the Biomedical Domain with Description Logics

URL: http://www.purl.org/goodod/guideline

Version 1.0
December 2012

Send feedback to:
martin.boeker@uniklinik-freiburg.de
ludger.jansen@uni-rostock.de

Schulz S[1,3], Seddig-Raufie D[1], Grewe N[2], Röhl J[2],
Schober D[1], Boeker M[1], Jansen L[2]

[1]: Institute of Medical Biometry and Medical Informatics,
University Medical Center Freiburg
[2]: Institute of Philosophy, University of Rostock
[3]: Department of Medical Informatics, University of Graz

11th December 2012

Contents

1. Introduction

1.1. Main Objectives and Intended Usage of this Guideline

Ontology engineering is mainly done by domain experts who are specialists in their domain but have, if at all, limited knowledge in logics, computer science, or analytic philosophy. The vast literature on formal ontologies in general or on biomedical ontologies in particular is not suited (nor intended) to serve as an educational resource that would help domain experts to become good ontologists. Existing educational resources focus rather on ontology tools and languages than on good practice. The ongoing controversies encompassing practical, philosophical and computational aspects are necessary and fruitful for researchers and theoreticians but are confusing and discouraging for engineers. The purpose of the GoodOD guideline is to pave the road for domain experts who want (or need) to become ontology engineers. It pursues the openly pragmatic approach that ontology engineers only need to know ontology theory to the extent which supports the (measurable) outcome of their work. In its main body it builds upon given philosophical and methodological principles as well as preexisting tools and resources. In particular, it builds on:

- (pragmatic) scientific realism,
- first order logics and its subset OWL-DL (description logics),
- set theory,
- an upper-level ontology with high-level categories and a canonic set of relations,
- constraints which limit the freedom of choice when building coordinate expressions,
- the ontology editor Protégé 4,
- a complete DL reasoner, such as HermIT,
- ontology design patterns, and
- naming conventions.

Given these precepts the guideline contains all knowledge to be acquired by a user to be able to solve ontology engineering problems in a foreseeable and non-arbitrary way. The didactic concept is to convey the skills not only by theory but also by practical examples. The guideline itself is not a tutorial, but it is designed in a way to facilitate the re-use of its content for the building of course materials (Boeker et al., 2012). Although we do recommended to read the

chapters in sequential order, readers with some acquaintance with the field can also read single chapters as stand-alones. While the recommendations of the guideline are independent from the software used, terminology and examples are drawn from the Protégé editor. Readers that are not acquainted with this programme should consult the appendix before reading section 4.3 or any subsequent chapter.

The spirit of the ontology engineering guideline can be outlined with the following comparison: To build good quality furniture from raw materials one needs to acquire the knowledge and skills of carpentry which would take several years of apprenticeship. But everybody is able to build pieces of furniture (with a predictable result) using pre-fabricated components. However, this requires intelligently constructed components together with intelligently designed manuals. The degrees of freedom are drastically limited but the result is achieved with considerably less effort. We pursue the same goal with this GoodOD guideline.

1.2. Structure of the Document

The following document is divided into five parts and a technical appendix. Depending on the reader's background some sections may be skipped, but they are best read in the order presented as there is progress from the more general information to the more specific and pragmatic. It starts in chapter 2 with a brief outline what is meant by "ontology" by philosophers and computer scientists, respectively, and how the distinction between an ontology and other structures of knowledge representation can be drawn. The main taxonomic structure and the basic elements, classes and relations of ontologies are introduced and explained. Chapter 3 introduces description logics (DL), the formal language of ontologies used in the guidelines and explains its usage in building ontologies. In chapter 4 top-level ontologies are distinguished from domain ontologies and intermediate-level "upper domain ontologies". The most important top-level category distinctions are outlined. The second part of this chapter describes BioTop, an upper domain ontology for the domains of biology and medicine, in considerable detail. The two remaining chapters focus on actual rules and guidance for building DL ontologies within the framework described. This starts with rather general principles for ontology design concerning criteria for what sorts of classes and relation should be admitted to an ontology and is followed by the description of concrete and application-centered ontology design patterns that can help with frequent problems in ontological modelling.

1.3. Typographical Conventions

In accordance with Protégé and BioTop, in this guideline we use:

- *italics*, upper case initial and camelback notation capitalization for class terms (like *Disposition*, *IndependentContinuant*, *UpperFemurOfTheLeftLeg*, *OxygeneMolecule*)
- **bold font**, lower case initial and camelback notation for formal relations or object properties (like **hasProperPhysicalPart** or **participatesIn**)

- a `fixed-width font` and lower case initial for logical connectors (like `and`, `or`, `not`, `some`, `only`).

1.4. Document Status and Feedback

This document was written as a collaborative effort within the GoodOD project, a joint research project of philosophers from the University of Rostock and medical information scientists from the University of Freiburg, funded by the German Research Foundation (DFG). It reflects the experience of developing ontologies as well as working with ontology engineers and teaching students in the theoretical and practical basics of ontology engineering. Many chapters in this guideline benefit from research that we documented in previous publications. E.g., chapter 2 draws on Jansen (2010), section 4.1 on Jansen (2008a) and section 4.3.2.1 on Jansen & Schulz (2011). More information about the GoodOD project can be found at `http://www.purl.org/goodod`.

In many aspects, this guideline is work in progress and we want to further develop and improve on these guidelines. For this purpose, we rely on feedback from experiences in the practical use of our guidelines, and we appreciate suggestions for improvements, clarification and amendments. Please address you comments to Martin Boeker (`martin.boeker@uniklinik-freiburg.de`) and Ludger Jansen (`ludger.jansen@uni-rostock.de`).

2. Fundamentals

This chapter provides a short overview on what ontologies are, why they are built and what their elements are. The word "ontology" was first used in philosophy as a name for the study of the most general kinds of being or general metaphysics. By now, the term has been adopted by computer and information science and acquired a different if related meaning there.

2.1. What Does "Ontology" Mean?

The word "ontology" was coined in the 17th century to name the philosophical discipline of general metaphysics. It derives from the participle of the Greek equivalent of the verb "to be", *on* ("being"), the genitive of which is *ontos* ("of being"), and *logos*, which in this context means as much as "study" or "science". While the name is modern, the discipline named has roots back in ancient Greek philosophy. Aristotle, who can be considered to be its founding father, called it "first philosophy", and his treatises came to be known under the title *Ta meta ta physica* ("Those behind the physical"), hence the name "Metaphysics". The introduction of the newly coined term "ontology" became necessary, because the discipline of Metaphysics became more and more variegated and various sub-disciplines emerged, and "ontology" was chosen as a new name for what was known since as general metaphysics, i.e. as the study of being as such, as Aristotle put it. In the twentieth century, then, Willard Van Orman Quine coined the crisp characterization of Ontology as the study of what there is (Quine, 1948).

Furthermore, the term "ontology" was also used metonymically by philosophers not only to name a certain scholarly discipline, but also a theory from that discipline, i.e. a set of claims about "what there is" - about the most fundamental classes of entites in the world. Or, as another philosopher has more recently put it: "Formal ontologies are theories that attempt to give precise mathematical formulations of the properties and relations of certain entities" (Hofweber, 2012). It is in this meaning that the word "ontology" acquired a plural ("ontologies") and was combined as well with the indefinite article ("an ontology") and with number words ("four ontologies"). From here, the term entered information sciences, as researchers, especially in Artificial Intelligence, thought it necessary to provide artificial agents with common assumptions for reasoning about their environment. For a while, the study of ontologies in information sciences proceeded independently from philosophy. It was only at the turn to the twenty-first century, that information scientists and philosophers started actively to collaborate. In 1998, the first conference on "Formal Ontology and Information Sciences" brought together researchers from both fields, and by and by the new discipline of Applied Ontology emerged. Witness thereof is the launch of a new journal by that name in 2006, the publication of a text-book in 2008 (Munn & Smith, 2008), and the establishment of the International Organisation for Applied Ontology (IAOA) as a scholarly association in 2009.

Thus, the philosophical discipline of ontology is the study of the most general classes or categories of everything that exists, their dependencies and relations. *An* ontology is a set of claims about these classes of beings and the relations among them. Let us now turn to the specific meaning of "ontology" in information science.

2.2. What Is an Ontology in Information Science?

As we said, the term 'ontology' became increasingly popular in computer science, where it was roughly used to denote information artefacts that order, standardize, and describe the kinds of objects that occur in a domain. There is no terminological consensus about what criteria exactly an information artefact has to fulfill in order to count as an ontology.

Very influential was the definition by Tom Gruber of an ontology as "an explicit specification of a conceptualization" (Gruber, 1993). This formulation is as crisp as mysterious, but Gruber is more elaborate at another occassion, where he writes: "An ontology defines (or specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain. The specification of an ontology takes the form of the definitions of representational vocabulary (classes, relations, and so forth) that provide meanings for the vocabulary and formal constraints on its coherent use." (Gruber, 1993). Other authors deem this definition too liberal. For them, not any specification of any conceptualization would do, but only "a *formally* explicit specification of a *shared* conceptualization" (Studer et al., 1998).

These restrictions seem to be obvious when it comes to information technical support of, say, the life sciences, for only formal specifications can be handled by computers, and only conceptualizations shared by the scientific community could be relevant for information technological support of the life sciences. Much more discussion centers around the question whether the representation of a conceptualization really is the ultimative goal of an ontology, or whether it is only instrumental in representing the general features of a pre-existing domain of reality. The latter approach is often dubbed "realism" as opposed to "conceptualism". In recent ontological engineering, there are strong advocates of realism (Smith, 2004), but also fierce critics of this approach (Merrill, 2010).

But again, when it comes to life sciences, we do aim at representing reality: There are real patients suffering real diseases that should be helped with efficient medical treatment. Thus, within these guidelines, we understand ontologies as representational artefacts that attempt 'to give precise mathematical formulations of the properties and relations of entities in a domain' [Stanford Encyclopedia of Philosophy, Art. Logic and Ontology]. More precisely, it is classes of entities that are relevant for us, as biology does not aim at describing individuals, but at stating general truth: it is a nomothetic ("law-stating") science, not an idiographic ("particular describing") science. Biology text books have, e.g., a chapter about the horse, but they do not have chapters about Fury or Black Beauty. Among the classes of entities that are relevant for, e.g., biology, there are countable things like cells and organisms, measurable things like amounts of substances etc., as well as qualities and processes, and, for the use of the scientist, also all those objects used for materially or virtually representing information (words, books, files, etc.).

2.3. What Is an Ontology?

In information sciences, ontologies are special information artefacts. There are various types of information artefacts. Computer programmes, for example, contain algorithms and commands that have to be executed by a computer to fulfil certain tasks. Databases, on the other hand, do not contain commands; they are not "executable". While computer programmes contain prescriptive information, databases are rather of a descriptive nature. A patient database in a hospital, e.g., contains a lot of descriptive information about the hospital's patients. Ontologies are also of a descriptive nature. As we will use the term in these guidelines, ontologies do not contain information about individual patients and their individual patient histories, but rather information about types: types of living beings, types of diseases, types of treatments and so on. That is, we arrive at the following defintion: *Ontologies are information artefacts that attempt to give precise formulations of the properties and relations of certain types of entities* (Hofweber, 2012).

Good ontologies are formal, explicit and adequate:

- Good ontologies are **formal**: The meaning of terms in an ontology is unambiguously defined to avoid misunderstanding and stated using mathematical axioms and definitions that enable automated reasoning.
- Good ontologies use **explicit specifications** to make domain assumptions explicit for reasoning and support human understanding of a domain.
- Good ontologies are **adequate** to the domain to be represented and thus have to reflect current scientific knowledge available about the domain to be modelled.

2.4. Why Do We Build and Use Ontologies?

It is a commonplace that we live in the age of information. Nobody can keep track of the masses of scientific publications in a single discipline, let alone survey the results of science in general. To make this knowledge more available to the scientific community, documentation scientists have developed large systems of thesauri (like MeSH, *Medical Subject Headings*, developed by the US based National Library of Medicine) or standardized terminologies (like SNOMED, the *Systematized Nomenclature of Medicine*, developed by the College of American Pathologists). But terminologies have to cope with problems deeply routed within human languages. There are, as already observed by the pre-socratic philosopher Democritus, synonyms, homonyms and anonyms: Some things have more than one term that denotes them, some terms can denote quite different things, and for some things there are no terms at all (or, at least, no terms at present). One strategy to avoid this problem is to set up ontologies instead of terminologies: No longer to catalogue the terms scientist speak about, but to make catalogues of the things they speak about using these terms.

The commonplace about the ever larger amounts of information is true in particular for the life sciences, as they deal with highly variegated and complex phenomena. There are millions of species, each of which has its peculiar repertoire of organs and traits, of genes, proteins

Table 2.1.: Example ontologies

Name	Primary use	Number of representational units
Gene Ontology (GO)	annotation of gene-related information	37392 [1]
SNOMED CT	medical documentation	295542 [2]
Int. Class. of Diseases (ICD-10)	medical statistics	12.161[3]
BioTop	top-domain ontology	380 [4]
Basic Formal Ontology (BFO 1)	top-level ontology	39 [5]

and pathways. We cannot help but store our knowledge about life phenomena with the help of scientific databases, i.e. with computer based knowledge systems.

But once databases for the different aspects of life (like anatomy, genetics, molecular biology) are set up, we do have silos for our knowledge, but these are separated silos: We cannot combine the knowledge of anatomists with the knowledge of geneticists. This is, at least in part, due to the techniques and standards used in coding the respective domains in these databases. In order to achieve interoperability between various information systems (and between different versions of the same database) we need certain standards, and these standards can, again, be provided by ontology.

2.5. What an Ontology Is not

2.5.1. Ontology vs. Knowledge Base

In our perspective, ontologies are restricted to represent only those parts of the knowledge of a domain that fullfil the following criteria:

1. They describe the characteristics that all individual objects of a kind (members of a class) have in common (universals).

2. They give the precise meaning of domain terms (i.e. what they denote in the world to be represented).

We emphasize, in accordance with A. Rector (2008), that the really 'interesting' domain knowledge, which is often probabilistic, hypothetic, or context-dependent, is *not* subject of an ontology. In order to avoid misunderstanding we recommend not to use the term 'knowledge base' for ontologies.

One can distinguish between four different classes of knowledge and the corresponding information artefacts for the representation of that knowledge (Schulz et al., 2009):

[1] geneontology.org, accessed 15.06.2012.
[2] UMLS, SNOMED CT (version 2011), accessed 15.06.2012.
[3] de.wikipedia.org/w/index.php?title=Internationale_statistische_Klassifikation_der_Krankheiten_und_verwandter_Gesundheitsprobleme&oldid=106616331, accessed 15.06.2012.
[4] http://purl.org/biotop/biotop.owl, accessed 15.06.2012, without bridge classes.
[5] http://ifomis.org/bfo/1.1, accessed 15.06.2012.

1. lexico-semantic representation - terminology, thesaurus
2. representation of classes of entities and the properties they have in common - ontology
3. what is typically true in certain contexts - background knowledge
4. knowledge about indidivuals - inventory, e.g. patient database

For terminologies it is customary and often sufficient to have informal representations in the SPO-form (Subject - Predicate - Object) that is also used in simple English sentences. Such a format is, e.g., used by the Resource Description Framework (Klyne & Carroll, 2004), that allows to express such subject-predicate-object triplets. A similar structure is exhibited by the Unified Medical Language System (UMLS) metathesaurus, where two concepts are linked by a relational expression. Examples of such triplets would be "Aspirin subclassOf Salicylate" and "Aspirin prevents Myocardial_Infarction". Note that the differences between these statements, one expressing a taxonomic relation and the other a probabilistic causal fact, are not reflected by the triplet format. A statement about language use (not about the entities of the domain) would also be expressed analoguously: "Hypothermia has_synonym Fever". If one wants to "ontologize" statements from such sources, care has to be taken, which ontological relations are meant by the relational predicate and if these can be expressed in a logically strict way at all. Ontologies in the stricter sense used in this document always respresent classes of entities of a domain of the real world.

"Background knowledge" according to Rector (2008) may comprise default knowledge, presumptive knowledge and probabilistic knowledge. Statements expressing these classes of knowledge are assumed to be "typically" true under "standard circumstances", but not universally true. Examples are "*Smoking* causes *Cancer*" or "*Hand* hasPart *Thumb*". The latter cannot simply be translated into the universal statement, that all instances of *Hands* have a thumb as a part, because there are pathological cases of hands that have lost the tumb or never had one. And the relationship between *Smoking* and *Cancer* is obviously not such that every instance of smoking (or even of the habit of smoking) always causes an instance of cancer. Rather there is a strong, but only probabilistic correlation between them, so the statement could either be taken to express relative frequencies in a population or a causal tendency of smokers to develop certain classes of cancer. These classes of knowledge cannot easily be expressed within the usual knowledge representation formalisms. Because these statements are usually not strictly universal, special care has to be taken, if one decides to incorporate some of it into an ontology. In many cases the commonly used descriptions logics (DL; cf. 3) are not expressive enough.

Knowledge about individuals is also not dealt with by ontologies in the strict sense, rather in data structures like patient data bases that may be called "inventories" (see the next section).

The recommendation is that one should not try to incorporate knowledge of classes 1 and 4 in an ontology as this will likely lead to misunderstandings and false inferences. Some background knowledge can be incorporated into formal ontologies, usually demanding rather complex expressions and value restrictions, so no general advice is possible in this case.

2.5.2. Ontology vs. Inventory

We defined ontologies as information artefacts that attempt to give precise formulations of the properties and relations of certain types of entities. Much knowledge, however, concerns individuals. "Black Beauty is a horse" is, however, not a proper ontological assertion according to our definition, because it concerns an individual horse. Such knowledge has to be administered in special databases about individuals, which we could dub "inventories". An important example for inventories are patient records, which are nothing else but recordings about individual treatments of individual diseases of individual patients. Although ontological relations are often defined with recourse to individuals, ontologies are primarily about classes of entities, not about individual entities. And though it is often reasonable to connect inventories with respective domain ontologies, both classes of information artefacts are independent from each other, and we will only focus on ontologies in these guidelines.

2.6. The Elements of Ontologies

Ontologies are information artefacts. They represent types of entities in a given domain. But they are to be distinguished from the domain they represent. As information artefacts, ontologies contain representational units (Smith et al., 2006). In particular, they contain representational units that denote classes of entities or classes (class symbols) and representational units that denote relations (relation symbols). While an ontology itself consists of assertions made by use of these symbols, an ontology is not about these symbols. Ontologies are about the classes and relations their representational units refer to. That is, they are about certain features of the world. In a similar vein, a novel consists of a lot of words, but it is not about these words, but about certain things that happen to the literary characters referred to by those words. In addition to representational units, ontologies may contain metadata, such as labels, textual definitions, commentaries or axioms.

2.6.1. Classes of Individuals

The things we encouter in the world, be it in the laboratory or in daily life, are all individual things (or, for short, individuals). But some individuals resemble each other in certain important aspects. E.g., they are all red, or water-soluble, or upright-walking rational beings. That is, they belong to certain classes of entities; they are instances of these classes. For example, Germany is an instance of the class *Country*, Fury is an instance of the class *Horse*, and Cicero is an instance of the class *Human*.

In these guidelines, we follow the conventions to capitalise and italizise class symbols, i.e. those representational units that refer to classes. (See Chapter 5.3 for a comprehensive overview of recommended naming conventions for ontology development.)

Classes come along in hierarchies. That is, classes may have superclasses and subclasses. *Mammal* is a subclass of *Vertebrate* and a superclass of *Horse*, etc. This is expressed by the Relation "subclassOf":

- *Mammal* `subClassOf` *Vertebrate*

- *Horse* `subClassOf` *Mammal*
- *Wild horse* `subClassOf` *Horse*

Normally, classes are defined in order to group individuals that belong together due to shared properties. (We will put more precision to this issue in later chapters.) Class definitions specify these properties. Typically, good definitions have the form "an F that is G". This scheme is normally ascribed to Aristotle (already hinted at in Plato). It is therefore called the Aristotelian scheme for definition, because according to Aristotle, a definition is always given through a close superclass (the *genus proximum*) and the specific difference (the *differentia specifica*), i.e. that feature that distinguishes the class that is to be defined from all other classes of that superclass. The classical example for this scheme is the traditional definition of *Human being* as "rational animal" (an animal that is rational). But the scheme can easily be extended to other areas:

- A square is a rectangle with four sides of equal length.
- A chronic bronchitis is a bronchitis that is prevalent for at least two years.

It is fundamental to distinguish signs and symbols from the things they signify. Paul has two legs, while his name "Paul" has no legs, but four letters. Often, these things get confused not only in everyday speech but also in scientific discourse, not only in computer science in general but also in ontological engineering in particular. A useful convention to avoid such use-mention confusion is to always use quotes when mentioning a word instead of using it. Thus: Germany has 80 million inhabitants and Germany is a country. "Germany", on the other hand, has seven letters but no inhabitants and "Germany" is not a country but the name of a country. And while "the first name of the discoverer of the special theory of relativity" (in quotes: phrase is mentioned!) is a string of twelve words, the first name of the discoverer of the special theory of relativity (without quotes: phrase is used!) is a single word, namely the name "Albert".

2.6.2. Relations

2.6.2.1. General Remarks

The formal relations between the entities of a domain provide the main structure of an ontology. We distinguish properties and relations. Predicates like "blue", "has mass of 1 g" correspond to the properties of individuals that are blue or have a mass of 1 g respectively. As pointed out above individuals sharing such a property form a class. Therefore properties are often called classes in formal ontologies.

Relations in the proper sense correspond to many-place predicates. In the sentence "The pen lies on the desk" the two-place predicate "lies on" corresponds to a relation between the pen and the desk. In "The small intestine is located between the stomach and the large intestine", the three-place predicate "located between" expresses the respective ternary relation between small intestine, stomach and large intestine.

For most purposes binary relations are sufficient. And as common DL versions only support these, we will restrict our discussion to binary relations (somewhat misleadingly called "Object Properties" in OWL and Protégé). As we are interested in relations that hold between classes, that is universally between all instances of two respective classes, we will not consider cases like the individual pen lying on an individual desk. We are only interested in *general* or *universal* relations. Formal ontological relations are *formal* in the sense that they are not a further addition to reality but connect entities to more complex segments of reality (Schwarz & Smith, 2008; Smith et al., 2005). They make basic ontological structures explicit. Some formal relations express the ontological dependencies between dependent and independent entities, which will be discussed below in chapter 4 ("Upper Level Ontologies"). Generally, a parsimonious use of relations is highly recommended. Whenever possible, only relations that are already established in the top-level ontology should be used.

2.6.2.2. Taxonomies

Taxomonies are the most important structures in biomedical ontologies, the "taxonomy tree" forms the "backbone" of an ontology. Taxonomies are hierarchies of classes structured by the `subclassOf`-relation (often also called "is_a"-relation) that expresses subsumption of a subclass under a superclass, e.g.:

- *Dolphin* `subClassOf` *Toothed Whale*
- *Toothed Whale* `subClassOf` *Mammal*
- *Mammal* `subClassOf` *Vertebrate*

and so on. Both relata of the subClassOf-relation must be classes. The most general class of a taxonomy tree is often called "root" and the classes at the lowest level, which do not have further subclasses, are the "leaves" of the tree.

Formal Features of the `subClassOf`-Relation

- Transitivity: any subclass is necessarily also subsumed by the superclasses of its immediate superclass. If A `subClassOf` B and B `subClassOf` C it follows that A `subClassOf` C. (If *Dolphin* `subClassOf` *Toothed Whale* and *Toothed Whale* `subClassOf` *Mammal* then *Dolphin* `subClassOf` *Mammal*.)
- Reflexivity: Every class trivially subsumes itself. A `subClassOf` A
- Antisymmetry: If A `subClassOf` B it is false that B `subClassOf` A unless $A = B$.

Another important and useful feature of taxononomic hierarchies is inheritance: Subclasses automatically inherit the properties of their superclasses. Properties asserted for the superclass do not need to be restated for the subclasses of this superclass.

Semantics of Relations

The semantics of the `subclassOf`-relation and other relations are explained with the help of the "**instanceOf**"-relation. The **instanceOf** relation is not a relation between classes like all

the other relations which we consider in the ontology, but relates individuals to their classes: "Germany **instanceOf** Country" [6]. Because its relata are instances and classes the relation is obviously irreflexive (nothing instantiates itself), intransitive (classes are not instances of classes) and antisymmetric.

Now we can define the `subclassOf`-relation between classes by means of universal quantification over their instances:

- *A* subclassOf *B* if and only if for all x, if x **instanceOf** *A,* then x **instanceOf** B

Because in natural languages the copula "is" is used to express ontologically quite different things one has to be careful to distinguish the precise ontological relations that capture the different meanings of the word "is" in the natural language. Among other things we express both the instantiation of a class and the subsumption of a subclass in this way: "Flipper is a dolphin" means "Flipper **instanceOf** dolphin". But "Dolphin is a mammal"and "Mitochondrion is a cellular component" express the `subclassOf`-relation: "*Dolphin* `subclassOf` *Mammal*", "*Mitochondrion* `subclassOf` *Cellular Component*".

2.6.3. Metadata

Like other software artefacts, an ontology may contain metadata about the classes and relations it contains. Administrative metadata may consist of information about when and why a certain entry entered the ontology, and editorial metadata captures who the author of the entry was, or when and why an entry was updated, and who authored the update. Sometimes it might also be desirable to refer to the source for certain facts that are being encoded in the ontology. In this case, the meta-data of an entry will contain references to scientific papers or an internet source. Information about the use of terms, their synonyms etc. are other important metadata, that are helpful for the understanding of human users but do not belong to a proper ontology.

The ontology editor Protégé, for example, allows for metadata in various foms:

- Annotations of the ontology as a whole, e.g.: backwardCompatibleWith, contributor, coverage, creator, date, format, incompatibleWith, language, priorVersion, publisher, rights, seeAlso, subject, versionInfo
- Annotations of single classes or relations: definition, deprecated, identifier, label, synonym

While many of these entries are predominantly for the human users, all of them could in principle also be used to support the automated processing of the ontology. Besides the metadata elements provided by the RDF model itself (Comment, ...) the most important metadata provider is the Dublin Core (DC) delivering a universal reuseable set of more than 20 metadata elements available as OWL annotation properties.[7] Although not recommended, one's own custom tailored annotation properties can be used to store proprietary metadata as well.

[6] Instances can be independent continuants (like Flipper, patient #456, the left kidney of patient #456), dependent continuants (the individual grey of Flipper's skin, the weight of patient #456) and processes (Flipper jumping through a hoop, the surgical procedure for removal of the left kidney of patient #456). Cf. chapter 4.1 on page 28 for details on this distinction.

[7] http://dublincore.org/documents/dces, 24.09.2012

2.7. A Formal Characterization of an Ontology

As we defined an ontology as an information artefact that represents classes of entities and the relations among them, an ontology needs to contain terms for the classes of entities, terms for the relations between the entities, and claims to the effect that the relations are asserted of some entities. These assertions yield a graph-like structure whose nodes represent entity classes and whose edges represent formal ontological relations between these entity classes. Such structures can be formally characterized (Bozsak et al., 2002). Typical formal ontological relations are the subsumption relation *is_a* and mereological relations like *part_of*. Thus an ontology (in the terminology of Bozsak et al. (2002): a "core ontology") consists of

- at least one entity class (but probably many more)
- at least one formal ontological relation class (but probably some more).

More formally, (the core of) an ontology O is a triple <E, R, I> with

- E a non-empty set of nodes; R a non-empty set of n-adic formal relations
- and I an interpretation function that assigns any n-adic relation from R an extension that may comprise n-tuples with elements from E.

For short, we will speak of "formal relations", as opposed to "material relations"; the latter term will be used for the relational entity classes of an ontology. Though some of the entity classes can be relation classes, the set of the entity classes and the set of *formal* relations of an ontology must always be disjoint. In contemporary ontologies, formal relations are normally dyadic relations because of the restrictions imposed on them through the description logic used in these systems. In principle, also relations of higher adicity could be formal relations. Formal relations may or may not have instances among tuples of entity classes, though from an engineering perspective there seems to be no point in introducing formal relations that will not be used in the end. But if we allow for non-instantiated formal relations, then the smallest possible ontology consists of exactly one entity and exactly one formal relation class with an empty extension.

3. Description Logics (DL)

3.1. What Are Description Logics?

Description Logics (DLs) (Baader et al., 2010) are languages for expressing ontologies in a structured, computer-interpretable and formally well-understood way. They form the core of the OWL 2 ontology language.

Ontology engineers thus face two tasks with regard to description logics:

1. They need to make explicit their knowledge about the domain to be modelled and to represent that knowledge in a formally stringent way.

2. As description logics have a limited expressability they need to adopt this description of reality to respect the limitations of the description logic dialect in use.

These tasks shouldn't be underestimated, but the benefit of being able to maintain large scale ontologies that support automated reasoning can easily offset the care and energy that needs to be invested in these tasks.

3.2. Description Logics Basics

A Description Logic (DL) allows the specification of a model of reality in terms of classes, relations, individuals, and the logical connections that exist between them.

In DL, a class is a set of individuals that share some properties. *WaterMolecule*[1] is an example of a class: There are many individual water molecules and each and every one of them can be thought of as a member of the class *WaterMolecule*. Some characteristics of an individual water molecule are not specific to that molecule but common to all members of the class (e.g. its molecular weight). These features can be expressed by statements about the class. For example, an apparent truism about water molecules is the subsumption that every water molecule is a molecule, which can be rephrased as the following statement about classes: "(*WaterMolecule* `subClassOf` *MonoMolecularEntity*)"

The relationships between different individuals in an ontology are represented by object properties and data properties. Object properties express relations between two individuals, and as such they can also be used to express information about the relationship between different kinds of individuals, i.e. classes. For example, **hasPhysicalPart** is an object property that can be used to relate a water molecule to an oxygen atom. Since for every water molecule there

[1] We follow the convention of typesetting class expressions in *CamelCase* (i.e. uppercase characters at word boundaries within the expressions) and *italics*, property expressions in **boldface** and logical constants in a `fixed-width font`.

exists an oxygen atom that it is part of, this information can be expressed by the following DL-Statement: "(*WaterMolecule* `subClassOf` **hasPhysicalPart** `some` *OyxgenAtom)*". According to the formal grammar of DL, such expressions are always set in brackets in order to avoid ambiguities in more complex expressions. We strictly adhere to this convention in this chapter, though often (as in later chapters) the outermost brackets are suppressed.

An additional construct are data properties. They can relate individuals to concrete values, like their weight or volume. For example, one could express information about the molecular weight of water in the following way:

WaterMolecule `subClassOf` **bearerOf** `some`
`(`*PhysicalMass* `and` **molecularWeightQuantityLocated**
`value ''18.01528''^^xsd:double)`

We will mostly ignore this type of construct.[2] In the following sections, ontology engineers will learn how to correctly employ the language constructs provided by Description Logics when designing ontologies.

3.3. Description Logics: Syntax and Semantics

As with every well defined logic, Description Logics are described by specifying both their syntax and their semantics. The syntax determines the 'alphabet' of symbols that are allowed and a set of rules to build complex expressions from the letters of the alphabet. The semantics of DL specify the precise meaning of the expressions that conform to the rules of the syntax. Different syntactical dialects however can be used without changing the semantics of the logic. OWL 2 defines, for example, four syntactic dialects that are fit to express the semantics of an OWL 2 ontology (Motik et al., 2009). The syntax presented here is a variant of the so called 'Manchester Syntax', which is designed to be very easy to understand by the user and is also used in the Protégé ontology editor.

3.3.1. Working with Classes

Basic Class Descriptions Modellers will make use of top-level ontologies to populate their ontologies with important classes.[3] The root of the taxonomy in every OWL ontology is formed by the class *Thing*, which encompasses everything in the domain to be modelled. All classes added by designers of top-level ontologies and by domain modellers descend from that root (i.e. everthing is a *Thing*). We will now discuss the tools available to modellers to add further classes with specified semantics as subclasses of those inherited from other ontologies.

[2] The reason for this disregard is twofold. On the one hand, present versions of OWL 2 restrict the use of XML Schema Datatypes to those defined in XSD (and a few OWL specific ones). On the other hand, the treatment of units (centimeter, kilogram, joule) attached to data properties as well as their conversion is undefined so that they are less useful than they would be in the general case.

[3] Tip for users of the Protégé ontology editor: The taxonomic relationships between classes are presented by Protégé in a tree like manner on the left side of the "Classes" tab.

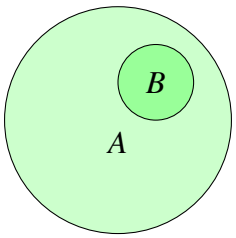

Figure 3.1.: (*B* `subClassOf` *A*)

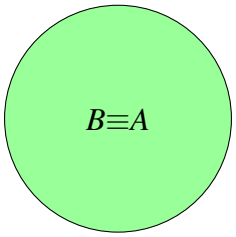

Figure 3.2.: (*B* `equivalentTo` *A*)

One major construct for this purpose is the `subClassOf` operator, which allows modellers to specify that every member of one class is also a member of the other (cf. Fig. 3.1). This means that every assertion that is true of the superclass (on the right side of the expression) is also true about the subclass (on the left side). For example, "(*WaterMolecule* `subClassOf` *MonoMolecularEntity*)" is compliant with this doctrine, i.e. there are no water molecules that are not at the same time mono-molecular entities.[4] When creating subclasses, modellers should always adhere to the principles specified in sections 2.6.2.2 and 5.4.

Usually it is even more useful to not only specify that a class *B* is a subclass of another class, but also to state the exact conditions of what it means to be a member of *B*. This can be achieved by specifying the conditions in a complex class expression and asserting that *B* is equivalent to the complex class. In Manchester syntax this reads: "(*B* `equivalentTo` *A*)". Semantically, both classes will always contain the same individuals (cf. Fig. 3.2). These classes are called "defined" or "equivalent classes".[5] In the class tree, classes that have this kind of definition are marked with three stacked horizontal lines ("≡"), which is the logical symbol for equivalence.

Two other important constructs to describe the relationships of a given class are the ''`DisjointWith:`''- and ''`DisjointUnionOf:`''-operators. The first allows modellers to specify that two classes do not overlap and hence do not share any members (cf. Fig. 3.3). This would be, for example, the case with the following statement "(*SubatomicParticle* `DisjointWith:` *MonoMolecularEntity*)". The ''`DisjointUnionOf:`''-operator is similarly useful because it

[4] Tip for users of the Protégé ontology editor: ''`subClassOf`'' statements can be added explicitly by adding superclass axioms but also implicitly generated by using the "Add subclass" icon to insert a new class below an existing one.

[5] Tip for users of the Protégé ontology editor: The name "equivalent classes" is also used in the class inspector of Protégé.

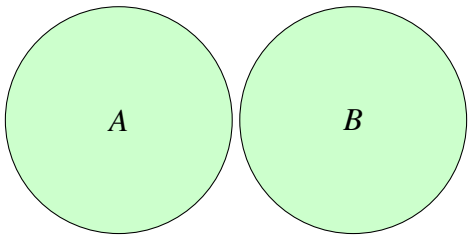

Figure 3.3.: (*A* `DisjointWith:` *B*)

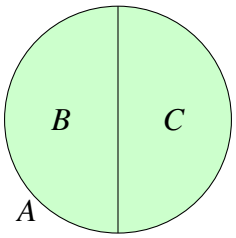

Figure 3.4.: (*A* `DisjointUnionOf:` *B*, *C*)

allows one to specify that a set of subclasses jointly exhausts their superclass while being pairwise disjoint with one another (cf. Fig. 3.4). For example, one would include the following assertion in an ontology: "(*SubatomicParticle* `DisjointUnionOf:` *Proton*, *Electron*, *Neutron*)" to specify that every subatomic particle is either a proton, an electron or a neutron, but not, say, a proton and an electron at the same time. For more information about this subject, please refer to section 5.4.1 on page 55.

Complex Class Expressions Indisputably, the facility to define equivalent classes is only useful if there is a way to generate complex class expressions. We will now turn to the basic facilities available for creating complex class expressions from more primitive ones. One of them is the "`and`"-operator, which takes two classes and uses them to build a new one that encompasses only those individuals which are present in both (cf. Fig. 3.5). For example the class (*EntireMolecularEntity* `and` *OrganicMolecularEntity*) would have all and only entire organic molecules as its members.

Likewise, modellers can create a new class by specifying it as the union of two existing classes: (*A* `or` *B*). Individuals from this class will either be members of the class *A* or of the class *B* or of both classes (cf. Fig. 3.6 on the next page). The corresponding operator is hence called "`or`". This operator has many important uses. For example, it is needed to specify that a class is exhausted by its subclasses (e.g. that all members of the class *A* only come from the subclasses B_1, or B_2, or B_3, cf. section 5.4.1 on page 55, this can also be expressed using the "`DisjointUnionOf:`"-operator). In other cases, modellers should be cautious about the use of the operator, because very often the constructed class will not correspond to a proper feature of the domain to be modelled.

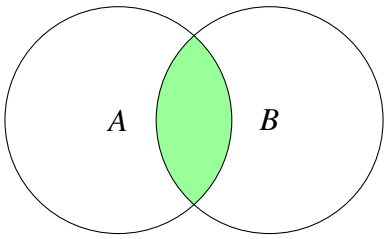

Figure 3.5.: (*A* `and` *B*)

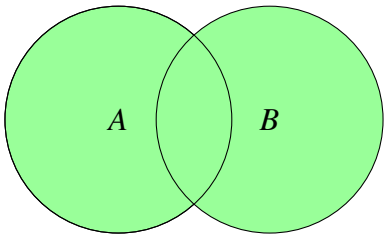

Figure 3.6.: (*A* `or` *B*)

Another construct that should only be used sparsely is the "`not`" operator. It can be used to construct classes that only contain those individuals which are not members of another class (cf. Fig. 3.7). For example, the class "(`not` *MonoMolecularEntity*)" would contain everything but monomolecules. Hence, all protons, horses, one dollar bills, and the part of London that lies south of the river Thames would all be members of this class. One can easily see that the individuals belonging to a class created by negation do not necessarily share any common features, an effect that is undesirable in a formal ontology. Modellers should only use such classes judiciously.

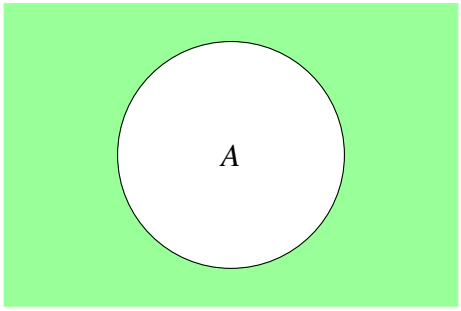

Figure 3.7.: (`not` *A*)

3.3.2. Working with Data and Object Properties

Top-level ontologies usually come with a variety of object properties that modellers can use. To avoid interoperability problems, these object properties should be used instead of defining many new ones for limited tasks. Thus modellers do not need to know how to create them, but they do need to be competent in evaluating their meaning.

Basic Property Descriptions Object and data properties specify the connection between individuals in an ontology or between individuals and extra-ontological, concrete values such as strings, integers, dates, etc. Every property has a *domain* and a *range*, which specify the classes the individuals connected by the property are members of. For example, the object property "**hasParticipant**" might have the class *Process* as its domain (because only individual processes have participants) and *MaterialObject* as its range (because only material objects can participate in processes). A property like this would have a Manchester syntax representation as follows:[6]

`ObjectProperty:` **hasParticipant**
`Domain:` *Process*
`Range:` *MaterialObject*

Properties can also be arranged in hierarchies similar to the hierarchy of classes by using the `subPropertyOf` axiom. For example, "**hasAgent** `subPropertyOf` **hasParticipant**" would declare that every pair of individuals connected through **hasAgent** is als connected by **hasParticipant** (but not necessarily the other way round), specifying that every agent is also a participant. Properties can also be described as `equivalentTo` and `DisjointWith:`, just as classes can be.

The definition of a property can be further enriched by specifying the characteristics of the property using the "`Characteristics:`" statement. This allows additional constraints to be placed on the property, for example that it is functional (i.e. for every individual in the domain, there is at most one individual in the range) or transitive. Transitivity one of the more useful property characteristics and allows modellers to state that if the property **hasProperPhysicalPart** obtains between a cell and its nucleus and between the nucleus and the contained DNA, the property also obtains between the cell and the DNA. For a complete list of possible property characterisations please refer to table 3.1 on the following page. Data properties only allow for the "`Functional`" characteristic.

For object properties, modellers can also specify the inverse of a property. For example, the inverse of **hasParticipant** would be **participatesIn**. Specifying the axiom "**hasParticipant** `inverseOf` **participatesIn**" means the following: The individuals of every pair connected by **hasParticipant** are, in reverse order, also connected by **participatesIn**. So if "tissue growth #12 **hasParticipant** cell #43" was asserted, "cell #43 **participatesIn** tissue growth #12" would also be asserted.

[6] Tip for users of the Protégé ontology editor: Properties can be added by using the "Add subproperty"-icon in the (object or data) property tab. The property description inspector will contain fields for specifying domain and range.

Charateristic	Definition
`Functional`	For every x, y_1, y_2, if $x\mathbf{R}y_1$ and $x\mathbf{R}y_2$ then $y_1 = y_2$.
`InverseFunctional`	For every x_1, x_2, y, if $x_1\mathbf{R}y$ and $x_2\mathbf{R}y$, then $x_1 = x_2$.
`Transitive`	For every x,y,z, if $x\mathbf{R}y$ and $y\mathbf{R}z$, then $x\mathbf{R}z$.
`Symmetric`	For every x,y if $x\mathbf{R}y$ then $y\mathbf{R}x$.
`Asymmetric`	For every x,y if $x\mathbf{R}y$ then not $y\mathbf{R}x$.
`Reflexive`	For every x,$x\mathbf{R}x$.
`Irreflexive`	For every x, not $x\mathbf{R}x$.

Table 3.1.: OWL 2 Property Characteristics

Complex modelling with properties can also include chains of object properties, where the connection between two individuals by means of a property can be expressed by a chain of other properties. For example, one might want to use the property **hasLocus** not only to specify the locations of material objects but also of processes. Since processes are located at the locations of their participants (e.g. the chess match is located where the players are located), this might be achieved by including the following statement in the definition of the property: " **hasParticipant** `o` **hasLocus** `SubPropertyOf` **hasLocus**", where the "`o`" is the operator for forming the chain link.

Building Classes using Data and Object Properties With help of some additional constructs, data and object properties can be used to generate complex class expressions. This is a powerful way to add additional axes of classification to an otherwise pure taxonomic ontology.

Since property expressions always take two partners (two individuals for object properties or an individual and a value for data properties), generating classes is done by fixing one of them. There is a number of constructs available to do this:

- **Value restriction:** Value restriction builds a class encompassing all individuals that are connected only to individuals of the class C by means of a given property **R** by writing "(**R** `only` C)". Since **R** corresponds to a verb phrase (e.g. "**hasParticipant**" or "**hasPhysicalPart**"), such expressions can be read quite alike. For example, the expression "(**hasProperPhysicalPart** `only` *SubAtomicParticle*)" might be useful when defining the class *Atom*.
  Modellers should note that the semantics of value restriction are such that it does not imply that an individual from the named class actually stands in the required relation for the property to hold. In our example it only specifies that, if an individual of the class has proper parts *at all*, they are from the *SubAtomicParticle* class. If one wants to require the existence of the individuals named, one has to employ additional constructs. For example, if a process is restricted by the statement "(**hasParticipant** `only` *Unicorn*)", this does not imply that unicorns exist.

- **Existential quantification:** Existential quantification can be used to define classes whose members require the existence of other individuals. For example, the class *Phos-*

phorylation will incorporate in its definition the claim to the existence of at least one phosphate molecule that participates in the process. This can be written as "`(`**hasParticipant** `some` *Phosphate*`)`". When using existential quantification, it is crucial to keep in mind that it commits one to the existence of additional individuals to the ontology. (Note the difference to value restriction.)

- **Number restriction:** Existential quantification has some more specific variants that allow modellers not only to specify *that* an individual exists but also the number of individuals. For example, to define what it means to be a water molecule, one must specify that it contains exactly two hydrogen atoms. This can be done with the "`exactly`" keyword followed by the number of individuals required: "`(`**hasPhysicalPart** `exactly 2` *HydrogenAtom*`)`". The keywords "`max`" and "`min`" can be employed in the exact same way. Hence, obviously, "`((`**hasPhysicalPart** `max 2` *HydrogenAtom*`) and (`**hasPhysicalPart** `min 2` *HydrogenAtom*`))`" would be equivalent to the previous statement.

- **Concrete values:** For data properties, the additional keyword "`value`" can be used to build classes that require a relation to a concrete value. For example, "`(`**molecularWeightQuantityLocated** `value ''18.01528''^^xsd:double)`" would describe the class of all individuals with a molecular weight of 18.01528 (e.g. water molecules). In this expression "`xsd:double`" describes the datatype of the value (in this case a double precision floating point number). The correct usage of such datatypes is, however, beyond the scope of this document.

3.4. DL Ontology Pitfalls

The Open-World Assumption

Characterised this way, DL ontologies exhibit a few features that lead to remarkable differences to other systems that are used to store structured information: In relational databases, for example, the absence of a fact from the database implies that this fact is false. This behaviour is usually described as *closed-world* assumption: A relational database assumes that it knows everything about the world; things it is not explicitly told about must thus be false.

An OWL 2 ontology on the other hand uses so called *open-world* semantics: If a given fact is not present in an ontology, it is simply not known and hence still might be true. This also means that the system will try to answer queries based on what is compatible with what it knows. For example, suppose an ontology contains the following class description.

`Class:` *OxygenMolecule*
`equivalentTo` **hasPhysicalPart** `exactly 2` *OyxgenAtom*

This class description might seem valid, but it is in fact too broad to capture what it means to be an oxygen molecule. While it succeeds in excluding ozone molecules to be classified as oyxgen molecules (since they consist of three oxygen atoms), it does not exclude any molecule that has exactly two oxygen atoms and an unspecified number of other atoms, e.g. nitrous acid (HNO_2). The reason for this is that having a nitrogen and a hydrogen atom as parts is

compatible with what the system knows about oxygen molecules. Additional statements need to be added to the class description in order to specify that an oxygen molecule consists of exactly two oxygen atoms *and nothing else.* For more information about this, please refer to 6.3.2 on page 66 which deals with an ontology design pattern called *closure*.

No Unique-Name Assumption

Another peculiar feature of DL systems such as OWL 2 is that they lack the so called unique name assumption: Two individual names may be shown to refer to the same object in the model. This is usually only a problem when dealing with individuals in an ontology, which modellers are advised not to do. For example, if one were to add an oxygen molecule called "Alice" to an ontology, and three oxygen atoms called "Bob", "Charlie", and "Dave", one would expect to derive a contradiction if each of them is said to be a part of Alice (because oxygen molecules only can only have two atoms). This expectation is mistaken: In this case, the reasoner will derive that either Bob and Charlie or Charlie and Dave are the same atom.

4. Upper-Level Ontology

There are various classifications of ontologies (Fig. 4.1). In this chapter, the following types are distinguished (Stenzhorn et al., 2007):

- A **top-level ontology** introduces general types (kinds, universals) and definitions that help unambiguously categorise the entities of the world into top level categories or classes. The number of top level categories is generally restricted and domain independent, such as *Function* or *Material Object*. Furthermore, most upper level categories provide a basic set of relations such as **partOf** or **hasQuality** together with axioms that restrict the use of these relations to entities of a certain kind *(Stenzhorn et al., 2007)*. Examples for top-level ontologies are BFO *(Grenon et al., 2004)* and DOLCE (section 4.2) *(Gangemi et al., 2002)*.

- An **upper-domain ontology** holds the essential core domain classes as an interface between both top-level and domain ontologies, like *Organism*, *Tissue* or *Cell* in the case of biology. An upper-domain ontology can also include more specific relations and further expand or restrict the applicability of relations introduced by the top-level ontology. An example for this kind of ontologies is BioTop (section 4.3) (Beisswanger et al., 2008).

- A **domain ontology** includes a multitude of low-level, domain-specific classes that comprehensively describe a certain (aspect of a) domain of interest, like, e.g., *Antisense RNA Transcription* or *DNA Replication* from the Gene Ontology.

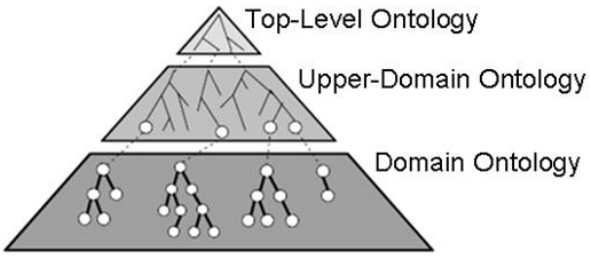

Figure 4.1.: The three levels of generality of a domain ontology

In the following, we will focus on classes that belong either to a top-level or an upper-domain ontology, as introduced above. Beginning with the question what, if anything, are the most general kinds of being we will discuss this question with reference to the categories of Aristotle, the first philosopher who is known to have asked this question. Starting from this, we will derive three basic dichotomies that can serve as principles to generate upper ontologies. We will then present two of the most important suggestions for top-level ontologies, i.e.

Table 4.1.: Aristotle's Ten Categories

Modern or Latin Term	English Translation/Thematic Question
essence	What is it?
quantum, quantity	How much?
quality	How is it?
relation	Related to what?
place	Where?
time	When?
position, posture	How is it situated?
habitus	Having, Owning
agere	What does it do? What actions does it produce?
pati	What does it suffer from? How is it acted upon?

the Basic Formal Ontology (BFO) and the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE), as well as BioTop, an upper-domain ontology for the biomedical domain.

4.1. What Are the Most General Kinds of Being?

4.1.1. Starting with Aristotle

When starting to construct an ontology from the top, we have to start with the most general kinds of being. In philosophy, such most general (and thus most basic) kinds of beings are called categories. The first to compile such a list of upmost categories was Aristotle (384–322 BC), the founding father of ontology (cf. Table 4.1). Aristotle names many of his categories according to the interrogative expressions one would use to ask questions whose answers would make reference to entities in the respective categories.

For his list, Aristotle seems to have proceeded on the basis of his experience in dialectical exercises and philosophical discussions. This explains the disparity of Aristotle's list of categories, the elements of which are not all of the same standing. There are two important ways in which Aristotle's categories fall into disparate groups. Some of them require a bearer without which they cannot exist, while others do not need a bearer but are themselves bearers of other entities; thus entities divide into dependent as well as independent entities (section 4.1.2). Moreover, some entities exist as a whole at every moment they do exist, while others stretch out in time; this divides entities into continuants as well as occurrents (section 4.1.3). These are already the two most important ontological dichotomies that can be used as principles of a top-level ontology. Finally we will introduce a third dichotomy, that is orthogonal to the other two: the distinction between universals (or classes) and individuals (section 4.1.4).

4.1.2. Dependent and Independent Entities

The criterion of ontological dependence is used by Aristotle to give special ontological status to the first category ("what is it?") in Table 4.1, so-called individual substances. Everything else in the list in Table 4.1 is dependent on the individual substances. (You cannot sensibly answer questions like "How much is it?","Where is it?", "How is it?" etc. unless you say of *what* these questions are asked.) Therefore the ten categories are not to be viewed as equals; rather, the individual substances are presupposed by the other categories. Customarily, the dependent categories are called accidents that *inhere* in the substances. A smile, a certain height, or a certain color always need a bearer to exist. It is not possible for someone to disappear and leave his smile behind. The height of a tree cannot continue to exist when the tree is destroyed. The color of a rabbit cannot remain in a room when the rabbit is taken out of the room. The smile, the height, and the color are dependent for their existence upon a bearer, a substance which has this smile, this height, or this color, among its properties. Let us summarize: Substances do not need the entities of other categories in order to exist, whereas the entities of the other categories require entities from the first category for their existence. For this reason substances are ontologically independent of accidents, while accidents are ontologically dependent upon substances. The notion of ontological dependence can be formally captured by the following criteria:

Definition of. *rigid dependence*: An entity x is rigidly ontologically dependent upon an entity y, if y could not exist if y did not exist, such as a person's body mass.

Definition of *generic dependence*: An entity x is generically ontologically dependent upon entities of the type F, if x could not exist if no entity of type F did exist. An example is a piece of literature. It cannot exist without the paper on which it is printed, or the computer memory where it is digitally stored. However, it does not have to be the *same* paper, or the *same* computer memory.

The group of dependent entities can be further divided into relational and non-relational entities. Relational entities are those that are ontologically dependent on multiple bearers, while non-relational entities are those that are ontologically dependent upon one bearer only (Jansen, 2006; Smith & Ceusters, 2007).

4.1.3. Continuants and Occurrents

Another important distinction can be made by observing that the Aristotelian categories "action" and "passion" differ in an important way from the others. Whereas a substance such as a bacterium, a quantity such as a length of 20 meters, or a quality such as red, exist as a whole at every point in time at which they exist at all, the existence of actions and passions is spread out over the course of some time interval. Whenever we encounter a bacterium, we encounter the whole bacterium at each point in time over the course of the bacterium's life. The process by which a bacterium reproduces, by contrast, takes place within time. The process of reproduction has a beginning and an end; it is composed of temporal parts, i.e. various phases that follow one another in time. By contrast, the bacterium has spatial parts – for example,

a DNA, a membrane, and a cytoplasm – which exist at one and the same time. Hence, we see that there are two kinds of entities that stand in intimate relation to one another, namely: (1) an organism and (2) its life or history. The organism itself is present as a whole at every point of its existence, while the life of the organism is spread out over multiple points in time. Entities without temporal parts which continue to exist through time like an organism we call continuants. Entities which unfold in time, that is, have different temporal parts, are things that *occur* in time and are called occurrents (Johnson, 1921). We need both continuants and occurrents in order to represent reality adequately.

If we picture the world at any single point in time, we will discover people, animals, artefacts, colours, sizes, and relations in our picture. But changes, processes, and events that are taking place at that point in time will not be visible. In order to represent these, we need a sequence of pictures instead of a single picture; we need a film. To obtain a complete picture of our ever changing world, we thus need two kinds of representation. On the one hand, we need "snapshots" of the world at individual points in time, which capture the continuants. Let us call such snapshots SNAP ontologies (Grenon & Smith, 2004). Included among SNAP entities are substances, quantities, qualities, relations, as well as the boundaries of substances, collections of substances, places such as niches and holes, and spatial regions such as points, lines, surfaces, and volumes. Additionally, SNAP ontologies comprise also the instantaneously existing instances of qualities and quantities which would otherwise be ontologically homeless. On the other hand, we need a representation of change, something like a film which represents entire time spans. We will call these SPAN ontologies (Grenon & Smith, 2004). Included among SPAN entities are processes and events, temporal regions such as time intervals with time points as their boundaries, as well as spatiotemporal regions. Time points, in spite of their lack of temporal extension, belong to the SPAN ontology and not to the SNAP ontology. A single SNAP ontology, which represents the world at a given point in time, is linked to this time point as to its date, but does not contain this time point as one of the entities in its coverage domain. Ontologies in information science may comprise only SNAP entities or only SPAN entities, but they may as well comprise entities from both categories.

4.1.4. Classes and Their Members

In addition to the two ontological dichotomies already discussed – independent vs. dependent entities, continuants vs. occurrents – there is also a third: that between universals and particulars, or classes and their members. The test for being a class is that the corresponding expression can be predicated of another entity. In contrast, particular entities that are members of classes, such as Socrates or my height, cannot be attributed to other entities. Sentences that contain as predicates the expressions 'is Cicero' or 'is my height' are not predications in the technical sense, but rather identity claims like 'Tully is Cicero' or 'Five feet is my height'. A general expression such as 'human' can appear both as the subject and as the predicate of predicative assertions, as in 'A human is a vertebrate', and 'Cicero is a human'. It is important to note that unlike in set theory, we do not have classes of classes: Classes and members are distinct, and no class is ever a member of another class. This contrast of members and classes has its foundation in the nature of things, and is not an arbitrary choice of the modeller.

Table 4.2.: The Ontological Square

	substantial not in a subject	*accidental, non-substantial* in a subject
universal, general Predicated of a subject	**III.** *substance universals* Human being Horse	**IV.** *non-substance universals* Being white Knowing
individual Not predicated of a subject	**I.** *individual substances* This human being This horse	**II.** *individual accidents* This individual whiteness This individual knowing

Taken together with the distinction between inhering and non-inhering entities, this yields a fourfold distinction of entities, the so-called ontological square (Figure 4.3). Universals correspond to general classes of objects in a domain and instances to the members of these classes. It is important to note that only "natural" classes correspond to universals, not arbitrary sets (cf. the design principles discussed in 5.4).

We need all four cells of the ontological square in order to capture the whole reality. This is in agreement with the commonsensical understanding of most people. In daily life, we assume that Barack Obama (field I) exists as well as the species elephant (field III), the virtue of courage (field IV), and the individual white colour of my skin, which ceases to exist at some time in summer, when my skin takes on a brown colour instead (field II). There are some basic relations that obtain among entities in the four fields of the ontological square:

- Individual accidents *inhere in* individual substances.
- Non-substance (accident) universals *characterize* substance universals.
- Individual substances *instantiate* substance universals.
- Individual accidents *instantiate* non-substance (accident) universals.
- Individual substances *exemplify* non-substance (accident) universals.

A picture of the world which did not provide a special place for occurrents would be incomplete. There are also important relations that obtain between occurrents and continuants, for individual substances take part in individual processes. We can thus expand the ontological square to an ontological sextet, which can be illustrated in Figure 4.2 (Smith, 2005). The relations of inherence, exemplification, instantiation, and participation govern the relations among the entities in these four fields.

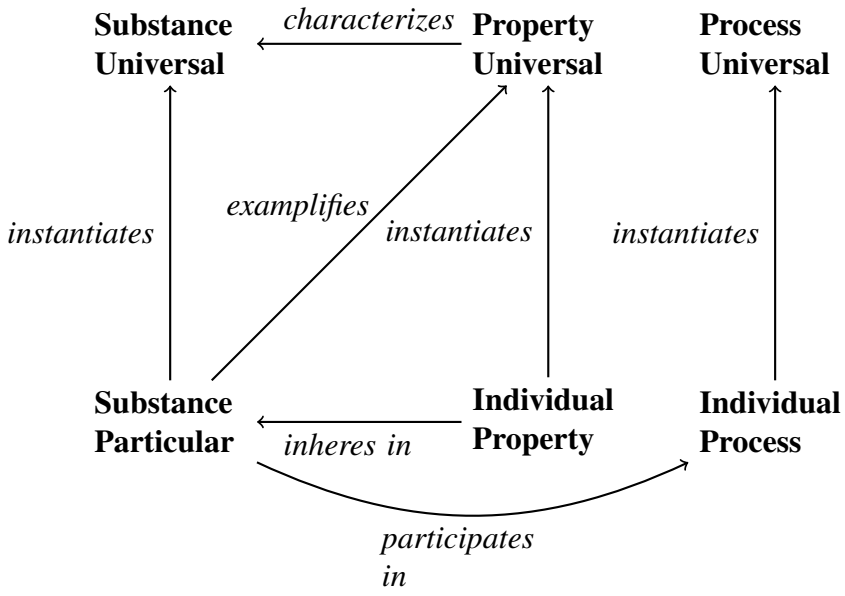

Figure 4.2.: The Ontological Sextet and Its Formal Ontological Relations

4.2. Two Important Top-Level Ontologies

4.2.1. BFO: Basic Formal Ontology

BFO, the Basic Formal Ontology, has been developed by the research group of Barry Smith at the Institute for Formal Ontology and Medical Information Science (IFOMIS), starting in Leipzig and continuing in Saarbrücken. BFO is a formal theory of the basic structures of reality. It is based on the Aristotelian background laid out in the preceding section. In fact, the basic principle of BFO is a "meta-ontological" combination of several ontologies. Among these are a series of SNAP-ontologies that represents the classes of continuants that are instantiated at certain points in time, together with a SPAN-ontology that contains all the classes of occurrents (Spear, 2006). (BFO is currently under further development; this section describes the current version 1.1. BFO 2.0 will come with additional classes and will also include formal relations.)

The main SNAP-categories of BFO are

- *Independent Continuant* with subclasses *Object*, *Object Aggregate*, *Site*, *Boundary* and *Part of Object*,
- *Dependent Continuant* with subclasses *Quality* and *Realizable* (with subclasses *Function*, *Role* and *Disposition*),
- *Spatial Regions* with subclasses in several dimensions: *Volume*, *Surface*, *Line* and *Point*.

The main SPAN-categories of BFO are

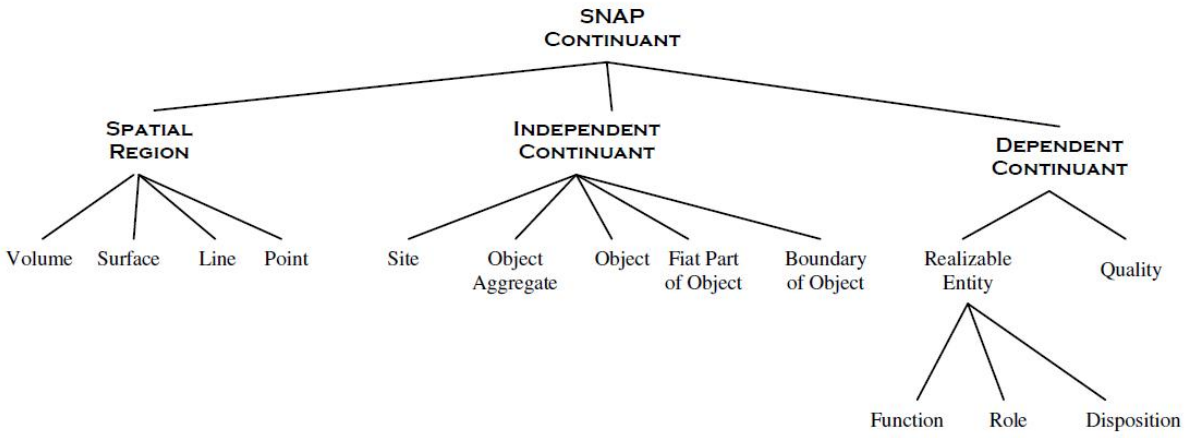

Figure 4.3.: The BFO SNAP Categories

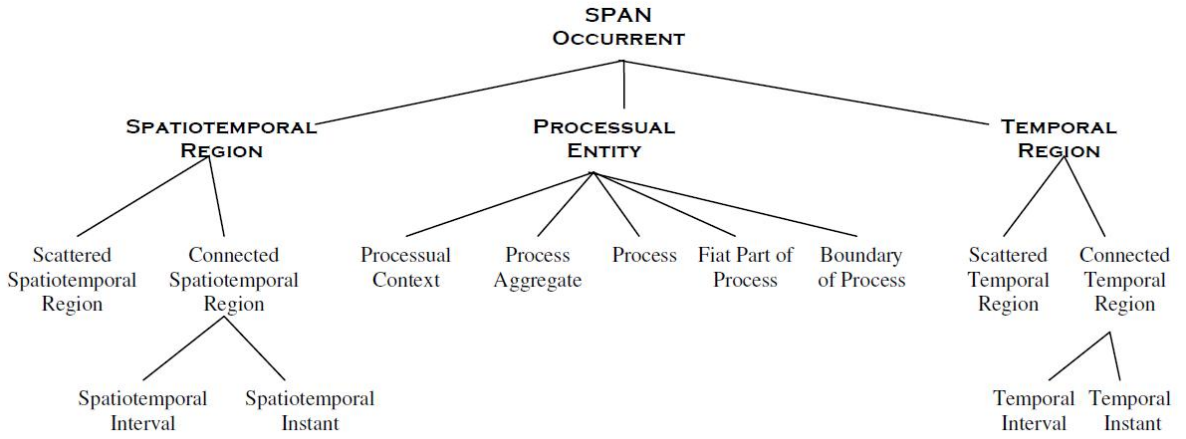

Figure 4.4.: The BFO SPAN Categories

- *Processual Entity* with subclasses *Process* as well as *Process Aggregate*, *Process Part*, *Processual Context* and *Boundary of Process*,
- *Spatiotemporal Region* with subclasses *Scattered Spatiotemporal Region* and *Connected Spatiotemporal Region* (with subclasses *Spatiotemporal Interval* and *Spatiotemporal Instant*),
- *Temporal Region* with subclasses *Scattered Temporal Region* and *Connected Temporal Region* (with subclasses *Temporal Interval* and *Temporal Instant*).

These categories are intended to be domain-independent, and anything included in a domain ontology should be subsumed under one of these top-level categories.

DOLCE	BFO
Endurant	SNAP-Entity
Perdurant/Occurrence	Process or Process aggregate or Fiat Part of Process
Physical Quality	Dependent Continuant
Temporal Region	Temporal Region
Spatial Region	Spatial Region
Physical Substantial	Independent Continuant

Table 4.3.: Comparison of DOLCE and BFO

4.2.2. DOLCE: Descriptive Ontology for Linguistic and Cognitive Engineering

The Descriptive Ontology for Linguistic and Cognitive Engineering, mostly named by its acronym "DOLCE", is a top level ontology aimed at capturing the ontological categories underlying natural language and commonsense. It has been developed by Nicola Guarino and his associates at the Laboratory for Applied Ontology (LOA) in Trento (Masolo et al., 2003).

DOLCE has a "cognitive bias", that is, DOLCE differs from BFO in not being committed to uncover the intrinsic nature of the world (which might be hidden from us), but only the overt structure of human thoughts and language. DOLCE is also based on a broadly Aristotelian background(Gangemi et al., 2002), though it deviates from BFO in several respects (Grenon, 2003). Some correspondences between BFO and DOLCE are listed in Table 4.3.

In spite of these striking similarities, there are also a number of differences between these two top-level ontologies:

- In BFO, there are no categories for abstract entities, nor for non-physical entities like mental or social objects.
- In BFO, there are no quality regions (values) qualities are located in. In BFO the quality regions from DOLCE are subclasses of a specific quality.
- In DOLCE, processes can have qualities, whereas in BFO not. (In BFO 2.0, modifiers of processes are being discussed to be included as so-called process profiles.)
- In DOLCE, all regions are abstract entities, even spatial and temporal regions.
- In BFO, there is no further sub-classifications of processes, which might be due to the fact that the criteria used for this purpose by DOLCE are highly language-dependent.

4.3. BioTop: An Upper-Domain Ontology for the Life Sciences

BioTop is an upper-domain ontology which is designed to support ontology developers with an ontological framework for the life sciences. BioTop is considered complete with regard to

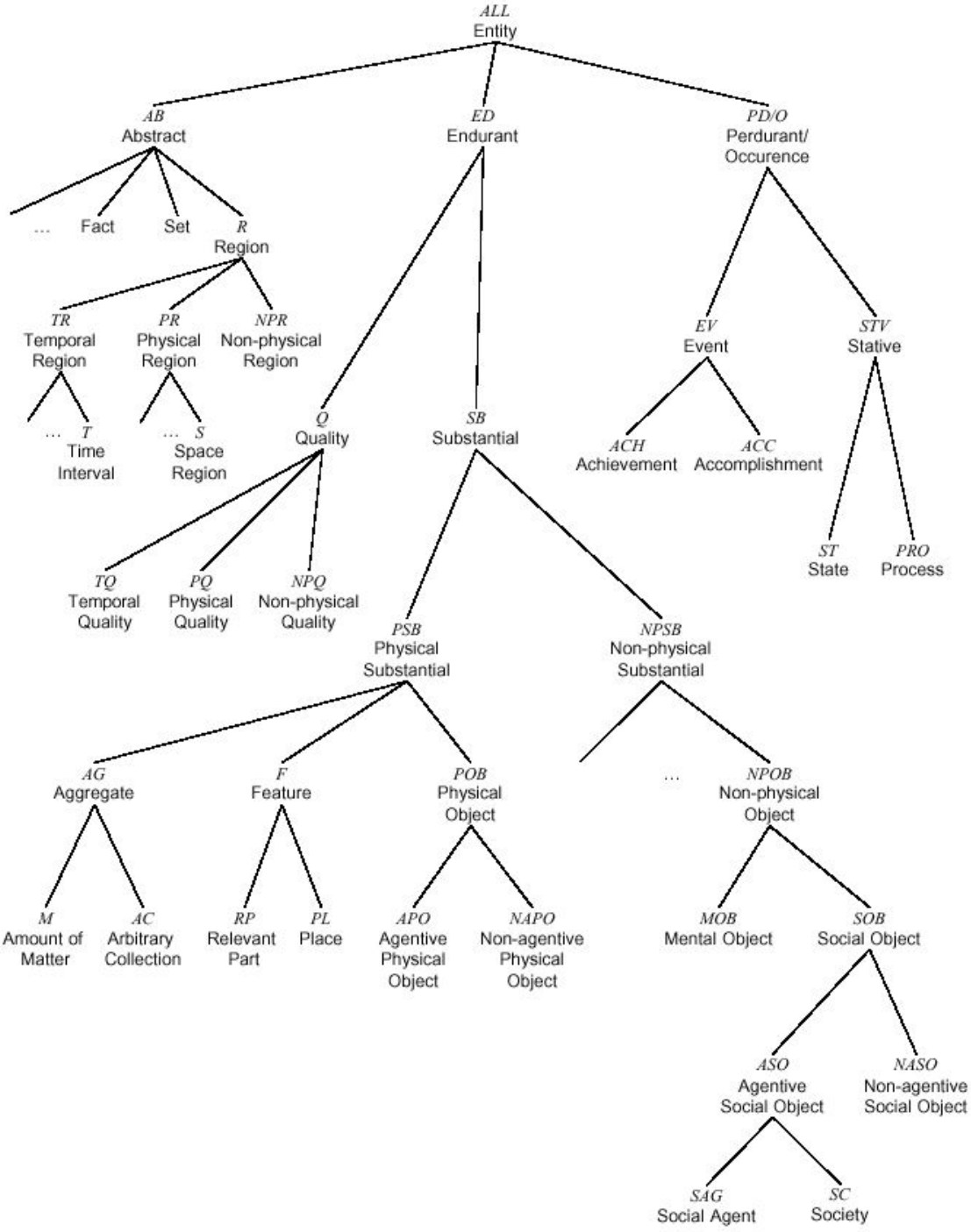

Figure 4.5.: DOLCE basic categories[1]

the set of given upper level classes and formal relations, so that developers only need to add subclasses to existing classes and to formulate axioms using existing relations.

BioTop classes and relations can be related to BFO classes as well as to DOLCE classes and relations when necessary. However, BioTop is primarily designed to be used as a standalone top-level ontology, and the labelling of its classes and relations is intended to facilitate its use. The upper level of BioTop is available as BioTop-Lite, a version of BioTop with a reduced set of classes. The expressiveness of BioTop is limited to OWL DL.

BioTop, Biotop-Lite and bridge files to import BFO and DOLCE as top level-ontologies can be downloaded from the BioTop website[2] where also documentation and related publications can be obtained. BioTop has been developed by the Institute of Medical Biometry and Medical Informatics (IMBI) at the University Medical Center Freiburg, Germany, and the Department of Computer Linguistics at the University of Jena, Germany. It is still under active development and is maintained by the Institute of Medical Informatics, Statistics and Documentation at the Medical University Graz, Austria, and the IMBI, Freiburg, Germany.

4.3.1. The Structure of BioTop

The general organization of BioTop follows the main principles of the top-level ontologies BFO and DOLCE. On the uppermost level, BioTop has only the classes *Particular* and *DEPRECATED*. The latter contains classes of former BioTop releases that have been retired, but which are preserved for downwards compatibility. Thus, the top-level class in effect is *Particular* with ten subclasses (Fig: 4.6). Of these subclasses, six are classes for continuant entities:

- *MaterialObject:* A material object is a continuant entity that has exactly one mass value and one volume at a time. However, material objects may have immaterial objects as parts, e.g. the heart has ventricles as a part.

- *ImmaterialObject:* All continuants of spatial relevance, such as points (zero-dimensional), lines (one-dimensional), planes (two-dimensional), and spaces (three-dimensional). They have an n-dimensional structure but no mass.

- *InformationObject:* Piece of information (not necessarily human), as it exists independently of any specific material carrier, such as the information contained in a book or in a computer file.

- *Disposition:* In contrast to qualities, a disposition is a realizable entity. This entails that it becomes manifest in some process. Which kind of process this is, depends on the physical make-up of the bearer of this process. The bearer of the disposition participates in the realized process as an agent. For instance, the fur colour of a mouse is not a realizable entity because it is just there and does not give rise to any processual manifestation. In contrast, an adult mouse's ability to reproduce is a realizable entity. The ability is there, but not necessarily its manifestation, i.e. the reproduction process that could take place for every instance of an adult mouse. A mouse has this ability to

[2] http://purl.org/biotop/, 21.09.2012

reproduce even if it never does; just as a glass has the disposition to break even if it never breaks.

- *Role:* In contrast to dispositions, a role is a realizable entity the manifestation of which brings about some result or end that is not essential to its bearer. This manifestation can be exhibited in some kinds of natural, social or institutional contexts. An example is a human's role of a student or a patient; or a molecule's role of a substrate or a product (of some (bio-)chemical process).

- *ValueRegion:* Value region is a spatial, temporal or abstract region in which values of qualities (see below) are located. Examples are the value 'green' as a value of colour, or the value '78 kg' of my body mass.

Two of the subclasses of *Particular* contain only occurrent entities:

- *Process:* A process is an occurrent. It has temporal parts. This means that not all of its parts are simultaneously present. Processes have material objects or immaterial objects as participants.

- *Time:* Time means a point or interval on the time axis, such as the 1st of January 2012, or the moment of an organism's death.

Two more subclasses of Particular may contain continuant as well as occurrent entities:

- *Quality:* A quality is an entity that characterizes some other entity but cannot exist independently of the former. An example is the body mass, or the green colour of a leaf.

- *Condition:* This class has been added as the union class of *MaterialObject, Process* and *Disposition* in order to represent the ambiguous nature of what is usually related as condition in the context of medicine. Here, many key terms denote entities of different kinds, such as 'tumor', which can be interpreted as a mass of malignant tissue, but also as a pathological process. Or the word 'allergy', which is used to denote an allergic disposition as well as an allergic reaction: Patients would say they are allergic to pollen even in winter when there are no pollen and they are not sick. The same wording will be used to refer to an acute allergic reaction in spring. The condition class therefore provides a place where classes can be put which represent the things ambiguous terms denote, without having do resolve the ambiguity.

All classes with the exception of *Condition* are mutually disjoint.

BioTop also introduces a set of relations. As BioTop uses many of the more expressive features provided by OWL 2, some significant restrictions need to be mentioned. All relations are OWL object properties, i.e. binary relations between individuals. Apart from the hierarchy-building SubClassOf-relation there are no relations between classes in BioTop. Expressions such as *ClassA* **rel** *ClassB* are syntactically wrong and semantically meaningless (see chapter 3 on Description Logics). Furthermore, ternary relations, such as **partOf** (a, b, t), with time as a third argument are not possible. Relational statements between continuants are therefore always interpreted as holding for any time. For instance, the axiom

Cell `subClassOf` **hasPart** `some` *CellMembrane*

is interpreted such that every member of the class *Cell* has *always* some member of the class *CellMembrane* as part.

The set of BioTop relations is considered exhaustive. It is strongly advised *not to introduce new relations* into the set of predefined ones, because the arbitrary and uncontrolled use of relations hinders the interoperability and comparability of different ontologies. BioTop relations are characterized by:

- Their hierarchical structure. Relations can have one or more parent relations. E.g., **hasAgent** is a subrelation of **hasParticipant**.
- The inverse relation they are connected with. E.g. **hasPart** has the inverse relation **partOf**.
- Their constraints. In most cases, their domain and their range are restricted by one or more BioTop categories. E.g., **abstractPartOf** has both its domain and range restricted by *Information entity*. Only information entities can be abstract parts or have abstract parts.
- Their algebraic properties. Some relations are transitive, e.g. **hasPart.**
- Relation chains. E.g., *A* **participatesIn** *B* and *B* **hasLocus** *C* entails *A* **hasLocus** *C*.

In the remainder of this chapter we will discuss important top-level classes from BioTop and detail how to use them when developing an ontology.

4.3.2. Material Object

Objective To represent everything which is material in the world and to relate it to other physical entities in regard to location and partonomy.

Top-level classes *MaterialObject, Atom, SubatomicParticle, PolymolecularCompositeEntity, StructuredBiologicalEntity*

Important relations The subrelations of **spatiallyRelatedTo**, in particular **hasLocus**, **locusOf**, **physicallyConnectedTo**

Description Material objects are continuants that have a mass. They can be atomic (*Atom*) or subatomic (*SubatomicParticle*), or single molecules (*MonoMolecularEntity*) which consist of at least two atoms which are bound to each other. All other material objects consist of two or more molecules (*PolymolecularCompositeEntity*). Material objects are related in terms of proper parts and wholes, by the transitive relations **hasProperPhysicalPart** and **properPhysicalPartOf**. Direct parts (components) that add up to a whole are linked by the non-transitive relation **hasComponentPart**, a subrelation of **hasProperPhysicalPart.** *StructuredBiologicalEntity* with its subclasses *Cell*, *CellularComponent*, *Organism* and *OrganismPart* is a subclass of *PolymolecularCompositeEntity*. An

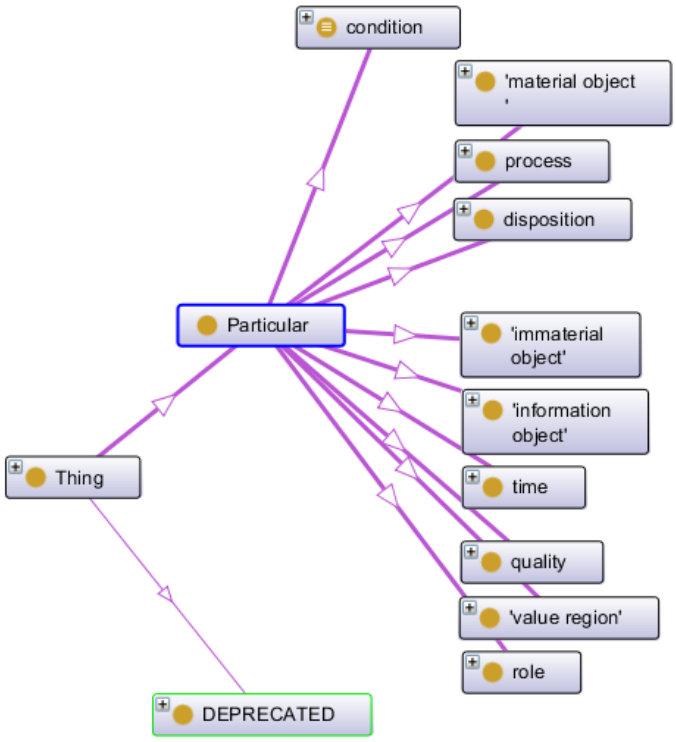

Figure 4.6.: The top-level classes of the BioTop upper-domain ontology.

organism is composed (**hasComponentPart**) of organism parts, e.g. individuals of class *Organ*. The parthood relation **hasProperPhysicalPart** is a subrelation of **locusOf**, the general physical location relation. Therefore, a biological entity which is part of an organism is also located in this organism (**hasLocus**). An organ part, e.g. a part of a muscle wall of the organ, is not necessarily a component part of the organism (no transitivity of the **componentPartOf** relation), but is necessarily proper physical part of the organism (due to the subrelationship of **componentPartOf** on **properPhysicalPartOf** and the transitivity of **properPhysicalPartOf**) and thus located in the organism (**hasLocus**).

Example A water molecule has proper physical parts hydrogen atoms and oxygen atom:

WaterMolecule `subClassOf`
(**hasProperPhysicalPart** `some` *HydrogenAtom*) `and`
(**hasProperPhysicalPart** `some` *OxygenAtom*)

If we consider that the only things that could be parts of atoms are subatomic particles, we can "close" this classes with a closure pattern (cf. 6.3.2) which leaves open the possibility for subatomic parts with the transitive relation **hasProperPhysicalPart** and fully define *WaterMolecule*.

WaterMolecule `subClassOf` *MonomolecularEntity* `and`

(**hasProperPhysicalPart** `some` *HydrogenAtom*) `and`
(**hasProperPhysicalPart** `some` *OxygenAtom*) `and`
(**hasProperPhysicalPart** `only`
(*HydrogenAtom* `or` *OxygenAtom* `or` *SubatomicParticle*))

4.3.3. Collective Material Entities and Compounds of Them

Objective To represent collections of material grains of the same sort or which is a heterogeneous compound of such collections.

Top-level classes *CollectiveMaterialEntity, AmountOfPureSubstance, PluralityOfOrganisms, CompoundOfCollectiveMaterialEntities*

Important relations **hasLocus/ locusOf**, **granularPartOf/ hasGranularPart**

Description In chemistry and biology it is important to distinguish between molecules and amounts of matter, or between single organisms and populations, respectively. Collective material entities (*CollectiveMaterialEntity*) are characterized as mereological sums of multiple grains of the same sort (e.g. a heap of pure sand). Because it is dependent on the knowledge of the observer, whether or not the grains are of the same sort, this class does not establish a clear categorial distinction, that is every material object can be seen as a collection of atoms or our heap of sand can be seen as a composition of different types of sand, (see below). Where this distinction is needed, *AmountOfPureSubstance* or *PluralityOfOrganisms* (subclasses of *CollectiveMaterialEntity*) can be used. The grains are granular parts of the collection (**granularPartOf/ hasGranularPart** which is also a non-transitive subrelation of **properPhysicalPartOf/ hasPhysicalPart**). E.g. a water molecule is a granular part of a portion of water. Even higher aggregations of collective material entities are compounds of collective material entities (*CompoundOfCollectiveMaterialEntities*) which have component parts of collective material entities. As such they are compounds of collections with no clear identity and unity criterion (e.g. an electrolyte solution on the basis of water which consists of a portion of water, portions of different metal atoms in ionic form and portions of different atomic and molecular anions like carbon dioxide or chloride).

Example A portion of water consists of water molecules as granular parts and portions of sodium chloride molecules consist of sodium and chloride ions. Because of the intransitivity of the relation **hasGranularPart** we can close the classes with a value restriction (Closure Pattern, cf. 6.3.2):

PortionOfWater `equivalentTo` *AmountOfPureSubstance* `and`
(**hasGranularPart** `some` *WaterMolecule*) `and`
(**hasGranularPart** `only` *WaterMolecule*)

An aqueous sodium chloride solution is a heterogenous compound with three ingredients, *viz.* water molecules, sodium and chloride ions. Again, due to the intransitivity of **hasComponentPart** we can introduce a value restriction as closure and fully define the resp. class.

AqueousSodiumChlorideSolution `equivalentTo`
CompoundOfCollectiveMaterialEntities `and`
(**hasComponentPart** `some` *PortionOfWater*) `and`
(**hasComponentPart** `some` *PortionOfSodiumIons*) `and`
(**hasComponentPart** `some` *PortionOfChlorideIons*) `and`
(**hasComponentPart** `only` (*PortionOfWater* `or` *PortionOfSodiumIons* `or` *PortionOfChlorideIons*))

4.3.4. Immaterial Object

Objective To represent what is physical but immaterial and to relate it to other physical entities in regard to location and partonomy.

Top-level classes *ImmaterialObject, OneDimensionalPhysicalEntity, TwoDimensionalPhysicalEntity, ImmaterialThreeDimensionalPhysicalEntity, PhysicalBoundary, Wave*

Important relations The subrelations of **spatiallyRelatedTo**, in particular: **hasLocus/ locusOf**, **physicallyConnectedTo, hasProperPhysicalPart/ properPhysicalPartOf, physicallyBounds/ physicallyBoundedBy**

Description Immaterial objects are continuants that have an n-dimensional spatial extension but, in contrast to material objects, no mass: lines (*OneDimensionalPhysicalObject*), planes (*TwoDimensionalPhysicalObject*), and three-dimensional spaces (*ImmaterialThreeDimensionalPhysicalObject*). Immaterial objects can have physical parts (**hasProperPhysicalPart**) which are immaterial objects of same or lower dimension (e.g., a cubic space (hexaedric or rectangular box) has six bounding rectangular planes as physical parts and one maximal cubus as a part). Material objects can have immaterial parts whereas immaterial objects cannot.[3] A natural cave is an object which has a wall usually of stone (*MaterialObject*) and a cavity (*ImmaterialObject*) as physical parts. This is pertinent in determining something which is in an object but is not part of this object (i.e. NOT **physicalPartOf**). A visitor of a cave is only located in the cave but not a physical part of it. However, we can imagine complex (distributed and not physically connected) material objects which have physical parts that are located in a cavity or a hole inside the complete object and are physical parts of the object (imagine for example a work of art with part of an object "flying in the center of a hole", or an physical experiment in a collider with electrons being accelerated insight the spatial ring of the collider which are physical parts of the mechanical arrangement of the experiment but not physically connected to the surrounding material ring).
Material objects are bounded by immaterial objects (**physicallyBoundedBy**) which are

[3] The mereology here employed allows for material objects to have immaterial parts. For most of the biomedical use cases and their respective users, this is the most natural and least complex representation of an inherently very complex matter. The authors are aware of other mereological systems which might be much more elaborate with respect to formal representation of topological relations of parts and wholes.

either planes, lines or points. A cell is bounded by the outer surface of the cell membrane which is a surface. A bone can have a ridge or edge which is bounded by the line on top of the edge and the two faces on either side of the edge by corresponding planes.

Example The surface of a portion of water is a two-dimensional immaterial object which bounds a portion of water (and only of water):

SurfaceOfWater `equivalentTo` *TwoDimensionalPhysicalObject* `and`
(**physicallyBounds** `some` *PortionOfWater*) `and`
(**physicallyBounds** `only` *PortionOfWater*)

4.3.5. Structured Biological Entities

Objective To represent which is a physical part of an organism and to relate it to other anatomical entities in regard to location, partonomy and further anatomical relations.

Top-level classes *PolymolecularEntity, StructuredBiologicalEntity, Organism, OrganismPart, Cell, CellularComponent*

Important relations The subrelations of **spatiallyRelatedTo**, in particular: **hasLocus/ locusOf**, **physicallyConnectedTo, physicallyBounds/ physicallyBounded**

Description As outlined in the two sections above, structured biological entities are material objects as well as immaterial objects and also compounds of both. On the macroscopic level the organism (*Organism*) has organism parts (*OrganismPart*) which can be further classified into organ systems, organs and organ parts, as well as tissues and cells. The axiomatic description of biological entities is not trivial: we have to account for living vs. dead biological entities, material derived from biological entities (e.g. wood), canonic vs. non-canonic entities (e.g. a mouse without a tail), in situ vs. in vitro entities (e.g. blood samples) as well as different developmental stages (embryo, fetus, adult,...) (Schulz & Hahn, 2007)

4.3.6. Process and Participation

Objective To represent occurrents and their participants as well as their mereological organisation.

Top-level classes *Process, Action*

Important relations **hasDuration, hasPointInTime, processuallyRelatedTo, hasParticipant, hasAgent, hasPatient, hasOutcome, participatesIn, agentIn, patientIn, outcomeOf, hasProcessualPart/ processualPartOf, precededBy/ precedes, hasLocus/ locusOf**

Description *Processes* are extended in time and last a period of time (**hasDuration**), or occur at a point in time (**hasPointInTime**) as instantaneous processes. A *Process* is not

instantiated until it is fully completed. A specific transport process in which an atom or a molecule is carried from one location to another (e.g. the intra-membranal transport of a sodium-ion from inside the cell to the extracellular space) is not instantiated until the specific transported substance has arrived at the target location and the transport has been completed. When the specific transport process has been interrupted it did not occur and was not instantiated. Processes have temporal parts (**hasProcessualPart**, **processualPartOf**), which follow each other sequentially (**precedes**, **precededBy**). Only for parts which are already completed an existential statement is meaningful (*A* **precededBy** `some` *B*) because about possible future process-parts one can only claim that they are instances of a certain class (value restriction, *A* **precedes** `only` *B*) but not that they will necessarily come into existence.

Processes have at least one participant (**hasParticipant**) which is a material object. A process participant can bear (**bearerOf**) different roles (*Role*) depending on the process it participates in. E.g., instances of class *Human* can be either *Patient* or *Agent* in a *Process* dependent on their behaviour as acting or passively receiving. This can be differentially represented by the subrelations of **hasParticipant/ participatesIn**, **hasAgent/ agentIn** and **hasPatient/ patientIn**. Another role a participant can play is being an outcome of a process like the product in a chemical reaction (**hasOutcome**). A *Process* is an *Action* if an *Agent* **participatesIn** it.

Process participants have to be strictly discriminated from the location (**hasLocus**) of the process. An inflammatory process of the fingernail has some fingernail as participant. This is in the first place also the location of the process, but it does not imply that an inflammation of the fingernail is an inflammation of the arm or the whole body. Whereas location is transitive, participation is not. An inflammation located in the fingernail is located in the hand (by transitivity of the **hasLocus** relation) but an inflammation located somewhere in the hand is not *necessarily* the same as an inflammation of the complete hand which would mean that the complete hand **participatesIn** the inflammation.

If a *Plan*, *Disposition* or *Function* is realized (**hasRealization**) then only by an instance of the class *Process* (see below). If a *Role* is realized (**hasRealization**) then only by an instance of class *Participant* (of an instance of class *Process*).

Example Medical procedures are therapeutic and diagnostic actions which have a medical professional as agent and a medical patient as patient. Medical professional and medical patient is a non exhaustive, non disjoint participation of humans based on roles (see below):

TherapeuticalMedialProcedure `subClassOf` *Action* `and`
(**hasAgent** `some` *MedicalProfessional*) `and`
(**hasPatient** `some` *MedicalPatient*)

An intramuscular injection into the deltoid muscle is a therapeutic action which results in the placement of a portion of a drug preparation for intramuscular injection into the deltoid muscle of the patient both of which are modeled here as participants. Although DL is not

sufficient to formally express identity, we can express at least that the outcome of the action is the placement (location) of a portion of the respective drug in a deltoid muscle.

DeltoidMuscleIntramuscularInjectionProcedure `equivalentTo`
TherapeuticalMedicalProcedure `and`
(**hasParticipant** `some` *DeltoidMuscle*) `and`
(**hasParticipant** `some` *IntramuscularDrug*) `and`
(**hasOutcome** `some` (*IntramuscularDrug* `and`
(**hasLocus** `some` *DeltoidMuscle*)))

4.3.7. Qualities and Their Values

Objective To represent attributes of objects and their values.

Top-level classes *Quality, ValueRegion, QbjectQuality, ProcessQuality*

Important relations **inheresIn, bearerOf, hasQuality/ qualityOf, hasObjectQuality/ objectQualityOf, hasProcessQuality/ processQualityQf, qualityLocated/ qualityLocationOf**

Description Most things have attributes like size, colour, shape, duration or length which are called *Quality* and are dependent on the objects that bear them. The objects are **bearerOf** (**hasQuality**) the respective quality and the quality **inheresIn** (**qualityOf**) its bearer. The quality can have values which are either quantitative or qualitative. In OWL numerical (quantitative) values are best assigned with data type properties to the individuals as numbers whereas qualitative values are located (**qualityLocated**) in a BioTop *ValueRegion*. Qualities and their value regions can be used to re-classify the taxonomy according to qualitative criteria.

Example A substance which can be applied in a therapeutic context can have different therapeutic effects, e.g. anaesthetic, antibiotic etc. This type of therapeutic effect attribute can be represented as a therapeutic effect quality (*TherapeuticEffectQuality*) which is quality located in the respective therapeutic effect value region (*TherapeuticEffectValueRegion*).

TherapeuticEffectQuality `equivalentTo` *Quality* `and`
(**qualityLocated** `some` *TherapeuticEffectValueRegion*) `and`
(**qualityLocated** `only` *TherapeuticEffectValueRegion*)

Different pharmacologically active substances can be classified according to the location of the effect quality or qualities they bear:

PortionOfPharmacoActiveSubstance `subClassOf` *AmountOfPureSubstance* `and`
(**bearerOf** `some` *TherapeuticEffectQuality*) `and`
(**hasGranularPart** `some` (*MonomelucularEntity* `or` *Atom*))

PortionOfPenicillinG `subClassOf` *PortionOfBetaLactam* `and`

(**bearerOf** `some`
(**qualityLocated** `some` *AntibioticTherapeuticEffectValueRegion*))
`and` (**hasGranularPart** `some` *PenicillinGMolecule*)

PortionOfAcetylSalicilate `subClassOf` *PortionOfPharmacoActiveSubstance* `and`
(**bearerOf** `some`
(**qualityLocated** `some` *AnestheticTherapeuticEffectValueRegion*))
`and` (**bearerOf** `some`
(**qualityLocated** `some` *AntiphlogisticTherapeuticEffectValueRegion*))
`and` (**bearerOf** `some`
(**qualityLocated** some *AnticoagulantTherapeuticEffectValueRegion*))
`and` (**hasGranularPart** `some` *AcetylSalicilateMolecule*)

4.3.7.1. Taxonomic Differentiation of Organisms

Objective To represent the hierarchy of taxa in the biological realm by location of a quality to respective values

Top-level classes *Quality, ValueRegion, QbjectQuality, TaxonQuality, TaxonValueRegion*

Important relations **inheresIn, bearerOf, hasQuality/ qualityOf, hasObjectQuality/ objectQualityOf, hasProcessQuality/ processQualityOf, qualityLocated**

Description Organisms are taxonomically ordered in a complex hierarchy of taxa which are represented by the location of the taxon quality (*TaxonQuality*) in a corresponding disjoint partition of hierarchically ordered taxon value regions (*TaxonValueRegion*). Hence, a taxonomical position of an organism can be assigned by the location of its taxon quality in the corresponding taxon value region. The logical soundness of this assignment can be proven and the super-/ subclass relations of an organism with regard to the taxonomic relations can be queried (Schulz et al., 2008).

Example An animal is an organism with a taxon quality which is quality located in the *KingdomAnimaliaValueRegion*. A vertebrate is an animal whose *TaxonQuality* is located in the *SubphylumVertebrataValueRegion*.

4.3.7.2. Differentiation between Canonical and Pathological

Objective To represent canonical and pathological objects and processes by location of a quality to respective values.

Top-level classes *Quality, ValueRegion, QbjectQuality, ProcessQuality*

Important relations **inheres in, bearer of, has quality/ quality of, has object quality/ object quality of, has process quality/ process quality of**

Description Anatomical structures as well as bodily states and processes can be either canonical or pathological. Canonical is defined as being in a state or condition in a normal

'range' of function, structure or behaviour whereas pathological is outside of this range. Only in few cases is the assignment of one of these two conditions based solely on quantitative and objective data but in most cases it is an attribution by a health professional who states this. Thus, it is best represented as a quality *Canonicity* which can be located in a value region which has exactly two values *CanonicalValueRegion* and *NonCanonicalValueRegion*. The latter can be subtyped by need, e.g. with *Pathological-ValueRegion*.

4.3.8. Information Object

Objective To represent information.

Top-level classes *InformationObject, Plan*

Important relations **denotes/ denotedBy, hasRealization/ realizationOf**

Description Information is essential in life and not neccessarily of human origin. An instance of *InformationObject* is dependent on a physical carrier (**bearerOf/ inheresIn**), but independent of a carrier with regard to its encoded content. Information does "mean something" which is represented by the relation **denotes/ denotedBy**. The particular which is denoted by the information (e.g. some protein structure) is existentially dependent on the denoting individual (in this example the encoding gene). The other way round, the denoting information is only existentially dependent on its carrier but does not need to be decoded: *"HemoglobinProteinSequence* `subClassOf` (**denotedBy** `some` *HemoglobinGene*)" but *"HemoglobinGene* `subClassOf` (**denotes** `only` *HemoglobinProteinSequence*)".
Plan is a subclass of *InformationObject* which defines a series of steps that must be fulfilled. A plan can only be realized (**hasRealization/ realizationOf**) by a *Process*. With respect to a possible realization plans resemble dispositions and roles (see below).

Example A treatment plan is an information object which usually inheres in some paper (on which it is written) or some electronic device in which it is stored. It describes the actions and their sequence which will are planned to be undertaken to succesfully complete the respective medical procedure. The plan exists independently of the planned procedure. The planned procedure is existentially dependent on the plan (although a unplanned action can be identical in the sequence of procedural steps without being a realization of the plan).

4.3.9. Roles and Dispositions

Objective To represent the realizable entities disposition and role.

Top-level classes *Disposition, Role*

Important relations **bearerOf/ inheresIn, hasRealization/ realizationOf**

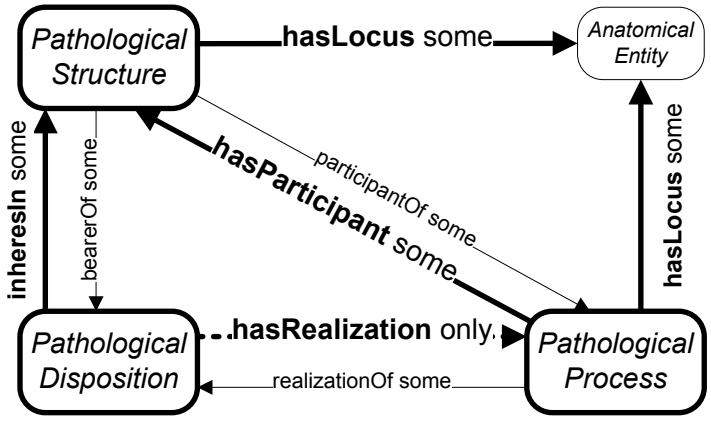

Figure 4.7.: Disease model (Scheuermann et al., 2009; Schulz et al., 2011)

Description Roles and disposition resemble qualities insofar they are also dependent enitities. They inhere in things (**bearerOf/ inheresIn**) which can realize them in processes (**hasRealization/ realizationOf**). A disposition inheres in some thing and can bring itself to existence in a respective process, e.g. the human disposition to speak is eventually brought to existence in most humans in the action of speaking. On the other hand, a role inheres in some thing and can be brought to existence by a certain type of process participation of the thing in which the role inheres, e.g. a human can have the role of a medical professional and a medical patient according to her participation in a medical procedure. Another example is the participation of substances in chemical reaction processes in which they can participate in very different roles, e.g. an enzymatic protein participates in its synthesis in the role of a translated outcome protein and in the reaction which it catalyzes as an enzyme. A thing can be bearer of multiple roles and dispositions hence a partition based on disposition or role cannot be disjoint. The realization of a disposition or role in an object is not necessary so that it can only be represented with a value restriction (**hasRealization** `only` *Process*) Röhl & Jansen (2011).

Example "Diseases" can be represented as a triple of *PathologicalStructure*, *Pathological Disposition* and *PathologicalProcess* (SDP triple) (Scheuermann et al., 2009; Schulz et al., 2011).

PathologicalStructure `subClassOf` *MaterialObject* `and`
(**hasLocus** `some` *AnatomicalEntity*)

PathologicalDisposition `subClassOf` *Disposition* `and`
(**inheresIn** `some` *PathologicalStructure*) `and`
(**hasRealization** `only` *PathologicalProcess*)

PathologicalProcess `subClassOf` *Process* `and`
(**hasParticipant** `some` *PathologicalStructure*)
(**hasLocus** `some` *AnatomicalEntity*)

To represent a specific disease like a heart valve disorder (mitral insufficiency due to mitral valve damage) and its consequences this ontological pattern is specialized according to the participating entities. In a successive patho-physiological chain, resulting damages can be represented as outcome of the pathological process of the respective patho-physiological stage.

MitralValveDamage `subClassOf` *PathologicalStructure* `and`
(**hasLocus** `some` *MitralValve*)

RetrogradeMitralBloodFlowDisposition subClassOf *PathologicalDisposition* `and`
(**inheresIn** `some` *MitralValveDamage*) `and`
(**hasRealization** `only` *RetrogradeMitralBloodFlow*)

RetrogradeMitralBloodFlow `subClassOf` *PathologicalProcess* `and`
(**hasParticipant** `some` *MitralValveDamage*) `and`
(**hasLocus** `some` *MitralValva*)

DilatingRetrogradeMitralBloodFlow `subClassOf` *RetrogradeMitralBloodFlow* `and`
(**hasOutcome** `some` *DilatedHeart*)

5. Good Practice Ontology Design Principles

5.1. Class Selection Principles

Not every word or phrase that is used in scientific discourse corresponds to a type that is fit for inclusion in an ontology. Modellers must exercise dilligence when deciding what classes are actually represented in the reality of the domain to be covered by the ontology. In what follows, we present suggestions on how to make an informed decision about the inclusion of a class in an ontology.

5.1.1. Linguistic Pitfalls for Class Selection

Disagreement with Reality

The goal of ontology engineering is to represent common, repeatable features of reality in the form of usable information artifacts. Consequently, the class expressions in an ontology are supposed to correspond to classes of entities in reality that share intrinsic features. We can refer to such classes through our linguistic expressions, but language cannot be taken at face value because it allows us to address arbitrary entities with a seemingly monolithic expression. Both "*Mitochondrion*" and *"Thing that fits into a* $1m \times 1m \times 1m$ *box*" can equally be used to refer to multiple entities, but they are not on equal footing ontologically because in the first case, instances share a great number of features, while in the later case they only share a contingent property such as size. Modellers have thus to avoid the following:

1. Introducing multiple classes that in fact refer to the same type of entity in reality: In the 19th century, for example, *general paresis of the insane* was considered to be a specific psychiatric condition, unrelated to any other disease. But later on, research has shown that it is just an advanced stage of a *syphilis* infection. Hence the predicates "is a syphilis infection" and "is a general paresis", which were thought to be unrelated, actually have common ground in reality and only one class, *SyphilisInfection* (which might have different stages) is needed.

2. Introducing only a single class that encompasses unrelated types. For example, the class *NileCrocodile* for a long time seemed to correspond to a single species of crocodiles, but has since been shown to comprise two unrelated species, *C. niloticus* and *C. suchus* *(Hekkala et al., 2011).* These classes should be used instead.

3. Introducing classes that do not have corresponding entities in reality: *Miasma*, *Unicorn*.

Closed Predicates

Additionally, at least some predicates can be readily disqualified by looking at the criterion of repeatability. Following David Armstrong (1980), we introduce the following terminology: We call a predicate *closed* if its semantics dictate that it can only be applied to a finite number of individuals. Examples include "*The first person to receive a heart transplant*" or "*One of the Three Stooges*". "*Variola virus*", on the other hand, will not be classified as closed. Even though there is only a limited number of variola virus particles in existence, there is nothing about the term "variola virus" that dictates that their number is necessarily limited. Such predicates which don't carry this type of inherent limitation will be called *open*. One can easily grasp that a closed predicate cannot be used to express general, repeatable features of reality. It is hence not useful to include classes corresponding to closed predicates in an ontology.

We thus derive the following rule for ontology development: If the applicability of a predicate P is restricted to a finite number of individuals by some inherent limitation, no class corresponding to P will be introduced to the ontology. Consequentially, modellers should never use the curly bracesn of set theory ("}", "{") to define classes using individual lists.

Impure Predicates

A second axis of classification for predicates is that of *pure* and *impure* predicates. We call a predicate *impure* if its definition makes essential reference to an individual entity (and *pure* otherwise). "*Person infected by patient #431*" would be an example of an impure predicate. Such predicates can be open predicates and hence be applied to a potentially infinite number of individuals. But since they make reference to some contingent fact, they convey little information about the *features* that individuals instantiating the corresponding class would have. It is thus not desirable to include them in an ontology. Sometimes predicates only make superficial reference to individuals, for example, one could have defined the Creutzfeldt-Jakob disease as "*The disease first described by Creutzfeldt and Jakob in 1920*." This use of individuals can be eradicated by replacing the reference to the origin of the description with the actual features of the disease described.

We thus derive the following rule for ontology development: If the definition of a predicate P contains essential reference to one or more individuals, no class corresponding to P will be introduced to the ontology. Consequentially, modellers will not use individuals in class definitions.

5.1.2. Further Class Selection Rules

No Metaclasses

Ontologies strive to describe one level of reality, hence all classes in the ontology should describe the underlying reality and not the vocabulary or classification criteria we apply to

them.[1] Hence classes like *MassTerm* or *EntityAlsoContainedInMeSH* will not be present in the ontology. One notable exception that is required to facilitate ontology maintenance is the inclusion of the class *Deprecated*, which subsumes all classes that were present in earlier versions of the ontology and need to be retained for backwards-compatibility.

No Ambiguous Classes

In scientific discourse, terms are often used ambiguously. The word "cancer", for example, can not only denote the ulcerous tissue in a patient (like in the sentence "He had his cancer removed") but also to refer to the disease process a patient is undergoing (like in the sentence "He is in the early stages of cancer"). If such ambiguities exist, disambiguating classes need to be introduced. Hence, modellers should not choose to include the class *Cancer* but rather *CancerousTissue* and *CancerProcess.*

No "Catch-all" Classes

Conventional medical nomenclatures, such as ICD-10, often include "catch-all" classes such as *FractureOfLegNotOtherwiseSpecified* to cover all cases that are not explicitly mentioned in the classification. Whenever possible, such classes should be avoided when designing ontologies because they do not convey additional information about the individuals instantiated by those classes. This requirement is, however, sometimes at odds with the requirement of exhaustivity, which will be discussed in section 5.4.1 on page 55.

No Epistemic Content

Since the goal of an ontology is to describe the underlying reality, epistemic notions should not be used to define new classes. Something is regarded as epistemic if it refers to our knowledge about the world rather than just the world itself. Hence classes like *ProbableCancer* have no place in an ontology. This also pertains to classes that give information about how a certain fact has been discovered, like *BacterialInfectionIdentifiedBySputumCulture.* Such classes are not admissible as well.

5.2. Specifying Class Metadata

5.2.1. What Is Metadata?

Metadata is data about other data that provides contextual information to increase the value of the data being annotated with it. For annotation of ontologies one needs to distinguish metadata that refers to the representational artefact (RA) as a whole (RA metadata) and metadata that refers to the representational units (RU) within a given representational artefact (RU metadata). On the RU level it is sometimes hard to decide what is metadata and what

[1] It is of course possible to create an ontology of classification systems or vocabularies. Such an ontology would declare the corresponding linguistic or conceptual domain to be the level of reality that it wants to describe. As long as this point is clear, no confusion can occur.

is data, since this depends on the granularity and richness of the given implementation/syntax. For example in the OBO Syntax the 'definition' metadata tag is implemented (hardcoded) as an RU within the OBO language, hence 'definition' is not regarded as metadata in OBO, but as an essential representational unit within the OBO Format. For the OWL syntax on the other hand there is no such RU and hence in this syntax this element would need to be formalized as and referred to as metadata. In most cases however metadata annotations have no formally defined semantics and are ignored by the automated reasoning programs.

Metadata elements can be drawn from standard providers or they can be implemented as proprietary owl:AnnotationProperties right in an ontology. OWL annotation properties can have either data values/literals (e.g. String or URI or rdf:XMLliterals (can embed arbitrary XML) or individuals as objects (e.g. the instance 'metadata incomplete' for the OBI:curation_-status property). Structured data within metadata annotation properties is usually defined in Backus Naur format.

5.2.2. Why Does One Need Metadata?

Enforcing a required set of minimal metadata to be delivered with an ontology will speed up human orientation and enrich searches within these resources. Metadata elements will serve as search attributes in query languages and will save the editors from performing more complex filter operations manually. Straightforward access to more granular contextual information will ease ontology administration, e.g. by providing access to editorial and development-history data. Curation status metadata will provide a means for semi-automatic evaluation and per-release checkups of our ontology, e.g. we can choose to release only terms with a curation status above a certain level. Ultimately the application of metadata will:

- *Increase data accessibility:* Metadata will increase the speed of data access by controlling and unifying the way data is accsessed, and specifying how and to whom what parts of the resource are available.
- *Increase data quality:* Facilitated access synergetically increases the data quality as it is continuously and frequently used and hence revised.
- *Increase data standardisation:* Rigorous usage of a metadata standard within an ontology will ensure compatibility among different data sources and different branches of the same representational artefact (RA).
- *Increase tools/application capabilities:* Data-consuming applications that utilise metadata will be able to interact more cleanly with each other. The quality of tools is increased by the deeper, machine-readable understanding of the underlying domain provided by the metadata.
- *Ease Ontology mapping and alignment:* Metadata capture will ease ontology comparison, evaluation and integration, e.g. through tools like PROMPT and for Ontology access portals like the NCBO BioPortal.

Metadata and ODPs

Metadata policies can be outlined as part of Presentation Ontology Design Principles (PODPs), specifically they are informed through the Annotation ODP[2], which provides annotation properties or annotation property schemas that are meant to improve the understandability of ontologies and their elements. Examples are the use of RDF Schema labels and comments which is crucial for manual selection and evaluation. In order not to re-invent the wheel and to keep development cost down, existing metadata schemes should be re-used. We here introduce some of the major existing metadata schema providers:

- *RDFS / OWL:* The RDFS and OWL languages themselves provide a few auditing and editorial metadata elements for RAs and representational units (RU), e.g. owl:versionInfo, owl:equivalentClass, rdfs:label, rdfs:comment, owl:DeprecatedClass. While the advantage of RDFS/OWL is their popularity in ontological communities, they are not sufficiently complete to be applied as the sole set of metadata elements for OBI. This, together with their lack of granularity, has led ontology groups to overload these elements, e.g., the rdfs:comment with editorial information as well as class definitions.

- *Protégé metadata ontology:* The Protégé metadata ontology (OWL-DL) provides some administrative metadata tags, e.g. Protégé:isCommentedOut to make the reasoner ignore some restrictions, and Protégé:todoPrefix to determine whether a property value is a "to do" item. The advantage of these is that they are well integrated with the Protégé editor environment and its reasoning facilities.

- *Dublin Core (DC):*[3] This well accepted standard provides RA and RU metadata descriptors for OWL-Full as well as OWL-DL. An OWL-DL version of the Dublin Core ontology is suggested to be using DC annotation properties to annotate our RA with context information such as ‘creator’, ‘date’, ‘language’, ‘publisher’, ‘title’ etc.

- *The OBI metadata scheme and potential OBO Foundry recommendation:* OBI has decided to formulate its own RU metadata scheme. [4] To overcome the limitations of the existing metadata schemes in terms of applicability to ontological RUs, OBI annotates its RUs (classes and relations) with a custom-built set of RU metadata elements. The reason was that not one of the existing schemes provided all the elements OBI needs. RA minimal metadata used in OBI Metadata that describes representational artefacts (RA) as a whole and that enables agents to determine whether a particular RA is suitable for their needs are called RA metadata. Valuable metadata elements are currently dispersed over different metadata standard bodies and no single standard provides integrated access to all metadata elements desired by OBI. This has led OBI to draw its RA metadata from four different metadata providers: Dublin Core, RDFS, OWL and Protege. The disadvantage of this ’mixing approach’ is the following: Because the standards available are not orthogonal, users importing more than one metadata scheme

[2] http://ontologydesignpatterns.org/wiki/Category:AnnotationOP, 21.09.2012.
[3] http://dublincore.org/, 21.09.2012.
[4] Cf. http://www.bioontology.org/wiki/index.php/Ontology_Metadata_Policy_%28from_OBI%29, 21.09.2012.

are left in constant doubt as to where to draw a certain metadata element from. RU minimal metadata used in OBI currently applies a set of self developed RU metadata elements. Compared to the Dublin Core, this metadata scheme has been shaped for a more ontology-centric coverage and is tailored to the group's needs.

5.2.3. Don't Get Stuck in the 'Meta-Ether'

Providing Metadata can easily get too expensive and time-consuming. The art is to avoid 'analysis-paralysis' and getting stuck in the 'Meta Ether' (since knowledge is fractal there is no limit to the level of detail/granularity a class and its metadata can be modelled). Metadata should not get too complicated to be used. For best compliance the metadata implementation should be applicable to a wide range of possible users and tools. Featuritis and scope creep have to be avoided when we want our scheme to be of general usability.

5.3. Naming Conventions

Efforts to create explicit typographic, syntactic and semantic concept labelling conventions have been carried out in isolation by most terminology developers. However, where naming conventions have been developed, widespread application has been hampered by several factors, most notably domain specificity, document inaccessibility and format dependency. To overcome this drawback, the OBO Foundry naming conventions effort has reviewed and compared existing naming conventions, distilled and published universally valid conventions applicable to the OBO and OWL formats. The development of their current set of consensus-based, reusable naming conventions was also informed by a survey carried out on sixty ontology developers. This is the categorisation of naming conventions as currently applied within the OBO Foundry set:

1. Be clear and unambiguous
1.1 Use explicit and concise names
1.2 Use context independent names
1.3 Avoid confusing and overloaded taboo words
1.4 Avoid encoding administrative metadata in names
2. Be univocous
2.1 Use univocous names and avoid homonyms
2.2 Avoid conjunctions
2.3 Prefer singular nominal form
2.4 Use positive names, i.e. avoid negations
2.5 Avoid catch-all terms
3. Reduce string variance
3.1 Recycle strings
3.2 Use genus-differentia style names
3.3 Use space as word separators
3.4 Expand abbreviations and acronyms

3.5 Expand special symbols to words
4. Align Typography
4.1 Prefer lower case beginnings
4.2 Avoid character formatting

For the full set including definitions and examples please refer to the OBO Foundry pages.[5] Note that, contrary to conventions 3.3 and 4.1 above, we use in this guideline uppercase beginnings for class names (to distinguish from the names for object properties) and the CamelBack type instead of space for word seperation, both in accordance with BioTop and Protégé.

5.4. Designing Taxonomies

Ontologies are not merely collections of unrelated classes. They also specify the relationships between those classes in a hierarchical structure, which is called a taxonomy. Ontological taxonomies identify the subsumption relationships between the classes in the ontology just as biological taxonomies identify the hereditary relationships between species, genera, ordines, etc. In an ontology, the subsumption links are specified by means of the so called 'is_a' relation which corresponds to the "`subClassOf`" construct. We remind ourselves again of the formal interpretation of this operator (cf. section 3.3.1 on page 19): class A subsumes class B ("`(`B `subClassOf` A`)`") if all individuals of class B are at the same time individuals of class A.

Often failure to comprehend the semantics of the subsumption relation leads to mistakes in ontology design. The following sections discuss possible problems that can arise when designing taxonomies and give directions on how to avoid them.

5.4.1. General Design Recommendations

The quality of a taxonomy can be vastly improved by taking into account a few general criteria (cf. (Jansen, 2008b). Two of them form the backbone for an adequate taxonomic classification:

Exhaustivity When adding classes to the ontology, new subclasses ideally exhaust their superclass. This means that every individual that falls under the superclass A also falls under one of its subclasses B_1,B_2,... B_n. The reason for this is that only the leaf-classes of a taxonomy do exist in a strong sense: There does not, for example, exist an individual that instantiates merely the class *LacticAcid* but neither *L-LacticAcid* nor *D-LacticAcid* since any existing molecule of lactic acid must be one of the isomeres.
Formally, the required constraint can be introduced by, for example, defining the class *LacticAcid* as `(`*CarboxylicAcid* `and (`*L-LacticAcid* `or` *D-LacticAcid*`))`.[6] The general principle here is that the superclass of the class in question is restricted by using the "`and`"-operator on the union of the subclasses.[7]

[5] http://obofoundry.org/wiki/index.php/Naming, 21.09.2012.
[6] Assuming that *CarboxylicAcid* is the direct superclass *(genus proximum)* of *LacticAcid.*
[7] Tip for users of the Protégé ontology editor: This constraint can be added to the "Equivalent classes" section under "Description" of the "Classes" tab.

Disjointness When adding classes to the ontology, new subclasses ideally do not overlap. This means that every individual that falls under the superclass A also falls under at most one of the subclasses B_1,B_2,... B_n. For example, no individual instantiating *LacticAcid* can instantiate both *L-LacticAcid* and *D-LacticAcid,* just as no individual instantiating *Eukaryote* can at the same time instantiate both *Plant* and *Animal*.[8]

If these two principles are adhered to, the resulting classification (superclass A and its subclasses B_1,B_2,... B_n) is said to be *jointly exhaustive and pairwise disjoint* (JEPD). This kind of classification ensures that every individual in the domain of interest falls under exactly one class. Ontology developers that are unsure about the concrete semantics of the OWL constructs needed to implement these suggestions should review chapter 3.

Additionally the following principles should also be adhered to when creating useful taxonomies.

Structuredness Good taxonomies usually exhibit a great deal of structure: They contain many levels of classes and subclasses, which increase the utility when compared with flat lists of classes. We can look to zoological taxonomy for an example: If the class *Animal* were to contain as its immediate subclasses classes like *Tiger, Mollusc, EurasianSparrowhawk,* and *Reptile,* it would be very difficult do gather any information from this taxonomy because it not only lacks structure but also misrepresents the status of the subclasses. *Tiger* and *EurasianSparrowhawk* are situated on the level of species, while *Mollusc* is a class on the level of phyla and *Reptile* on the level of (biological) taxonomic classes.

Organizing these classes in a proper hierarchy facilitates retrieving knowledge about entities on many levels, be it species, genus, phylum, or regnum. Users should avoid both introducing subclasses that exhibit different degrees of generality and skipping too many levels when dividing the topic domain. The depth of the hierarchy can, however, be limited by practical considerations.

Systematicity For every division of a superclass into multiple subclasses, coherent criteria should be applied. For example, a possible classification of the class *Molecule* could include (among others) the subclasses *BioMolecule* and *SilicateMolecule*. This classification is to be rejected because it uses incoherent principles to generate the subclasses: While *BioMolecule* would need to be defined by provenance (a biomolecule is a molecule that can be produced by a living organism), *SilicateMolecule* would be defined by chemical structure (a silicate molecule is a molecule that includes silicon bearing anions, such as $[SiO_4]^{4-}$). This incoherent way of classification can also lead to classes which are not disjoint: At first sight, *BioMolecule* and *SilicateMolecule* might appear to be disjoint, but further investigation or simply the broadening of the intended domain might reveal cases of silicates that are generated by biological organisms. Instead, users should divide classes in a principled way, e.g. taking into account only the chemical structure.

[8] Tip for users of the Protégé ontology editor: This constraint can be achieved by adding the classes that a certain class is disjoint with to the "Disjoint Classes" section under "Description" in the "Classes" tab.

5.4.2. Subsumption Misuse Problems

Modellers sometimes tend to use the subsumption hierarchy to represent relationships that are not really subsumptions. This leads to false and often misleading interpretations of the ontology, which are especially harmful if the ontology is used for automated reasoning. The confusions are largely due to the way we use the verb "to be" in everyday discourse. The verb "to be" is in fact used to express the subsumption relation, as in "An elephant is a mammal", which is the non-technical translation of a `subClassOf` assertion. But "to be" is also commonly used to express other information, for example instantiation, as in "Fido is a dog", or attribution as in "Grass is green". Usually, these cases of subsumption-misuse can be disambiguated by introducing the correct relation:

- Subsumption instead of instantiation: While '*Human* `subClassOf` *Mammal*' is a correct subsumption relation (every human is also a mammal, but not necessarily vice versa), '*Paul* `subClassOf` *Human*' is incorrect. Paul is not a subclass of *Human* (hence repeatably instantiable by different individuals), but an individual that instantiates the class *Human*. Otherwise it would be possible for different individuals to instantiate the class *Paul*.[9] Since ontologies are not supposed to contain individuals, this type of mistake should, a forteriori, never occur.

- Subsumption instead of parthood: The 'is_a' relation can also be confused with 'part_of': '*M-Membrane* `subClassOf` *Mitochondrion*' is not correct, because every m-membrane would at the same time be a mitochondrion. In fact, "`(`*M-Membrane* `subClassOf` **physicalPartOf** `some` *Mitochondrion*`)`" and not a subclass of it.

- Subsumption instead of attribution: Subsumption links like "*Strawberry* `subClassOf` *RedThing*", while being technically correct (at least every prototypical strawberry is also a red things), should be excluded from good taxonomical hierarchies because being red is nothing that unifies the classes it subsumes in a useful way. We would have to include not only strawberries, but also porphyry, blood, and a certain kind of card used by football referees as being subsumed by *RedThing*. It is more useful to describe the relationship as that of an object having a quality instead: "`(`*Strawberry* `subClassOf` **bearerOf** `some` *RedColourQuality*`)`".

- Subsumption instead of composition: One might be tempted to include the subsumption link "`(`*River* `subClassOf` *Water*`)`" in the taxonomy. This is clearly wrong because the two classes are individuated and identified quite differently: Water is the divisible stuff that can be portioned into glasses, buckets, or rivers. A portion of water remains the same portion as long as it contains the same molecules. A river, on the other hand, is a geographical formation that is *composed* of water, and as such, it exhibits a higher degree of unity than water: It cannot be arbitrarily divided into several rivers, whereas portions of water can be arbitrarily divided into further portions. Also, the bulk of water that constitutes the river can change while the river can stay the same. Hence, modellers should write "`(`*River* `subClassOf` **hasComponentPart** `some` *Water*`)`"

[9] Again, it is important not to confuse the name "Paul", which can, albeit ambiguously, apply to multiple persons, with the person Paul, who is a *single* human being.

5.4.3. OntoClean Taxonomy Design Principles

There is a number of additional criteria that could be applied to determine whether a subsumption link in the taxonomy is well founded. Some of these are specified in the OntoClean methodology (Guarino & Welty, 2009), which assigns "metaproperties" to classes in order to determine whether they can subsume one another. For example, OntoClean suggests that classes whose individuals differ in their unity criteria should not appear in a `subClassOf` assertion. For example, *PortionOfOxygen* and *OxygenMolecule* impose quite different unity criteria. Oxygen molecules are clearly delineated, countable entities while portions of oxygen lack such criteria. For everyday usage, OntoClean metaproperties are usually too complex but experienced modellers might find it valuable to consider them.

5.5. Relations for Rich Class Definitions

Usually rich ontological modelling involves more than just the creation of a taxonomy of classes. It also needs to represent the complex relations of entities from different classes. This is usually done by creating "`equivalentTo`" (full definitions) and "`subClassOf`" axioms that assert that instances of the class always stand in a certain relationship to other entities. This is done with complex class descriptions involving formal relations or, in OWL lingo, object properties (cf. section 3.3.2 on page 23). We will illustrate some specific problems of such definitions by taking the parthood relation as an example.

5.5.1. Example: Parthood Relations

One common type of relationship that is frequently used to express the spatial or temporal structure of entities is the parthood relation. Variants of the '**partOf**' relation are used when an instance of a class is a part of an instance of another class, e.g. the mitochondrion is a part of the cytoplasm, the index finger is part of the hand. So we have the relation *Mitochondrion* `subClassOf` **properPhysicalPartOf** `some` *Cytoplasm, Index Finger* `subClassOf` **properPhysicalPartOf** `some` *Hand*.

We can admit both spatial and temporal parts in our ontology. The **partOf** relation can also be applied to occurrent entities, i.e. processes. Processes can have temporal parts, that are partial processes or "phases". The diverse phases of cell division (mitotic phase, interphase, cytokinesis etc.) are all parts of the whole cell cycle process, hence *CellCycle* `subClassOf` **hasTemporalPart** `some` *Interphase*. We will not discuss parts of processes further, but it is important that the part_of-relation between processes is not confused with the "**hasParticipant**"-relation that holds between processes and the continuants that participate in these processes. E.g. *CellDivision* `subClassOf` **hasParticipant** `some` *Cell*.

As continuants may change over time with respect to their parts (or other entities they are related to in one way or another) one has to keep in mind that object property assertions or restrictions in an OWL ontology imply permanent relatedness. If it is asserted that *WaterMolecule* `subClassOf` **hasProperPhysicalPart** `some` *OxygenAtom* then water molecules must always have oxygen atoms as parts. On the other hand, it would be incorrect to state that

Tree `subClassOf` **hasProperPhysicalPart** `some` *Leaf*. This is because some trees have their leaves in summer and autumn, but not in winter.

5.5.2. Difficulties with Inverse Relations

When looking at axioms like *"WaterMolecule* `subClassOf` **hasProperPhysicalPart** `some` *OxygenAtom"* we observe that it includes an implicit universal quantifier ("for all water molecules ...") followed by an existential quantifier ("some oxygen atom"). This is called an "all-some-structure". All water molecules have some oxygen atom as parts, but it can *not* be deduced that every oxygen atom must be part of some water molecule. Sometimes that is indeed the case, but it does not simply follow from the logical properties of the relation. Another example on the contrary is *HumanTesticle* `subClassOf` **properPhysicalPartOf** `some` *HumanBody*. Every human testicle is part of some human body, but not every human body has a testicle among its parts. In this respect **hasProperPhysicalPart** and **properPhysicalPartOf** are inverses of one another. An inverse relation Rel^{-1} for the two-place relation R is defined as the relation that obtains between the relata, if they are exchanged: $A\ Rel\ B = B\ Rel^{-1} A$. Since the "all-some-structure" does not automatically transfer to the inverse, modellers need to carefully consider which of the two relations should be used.

6. Ontology Design Patterns (ODPs)

6.1. What Are Ontology Design Patterns (ODPs)?

Often, ontology developers encounter similar modelling problems in different situations. Because of this, **O**ntology **D**esign **P**atterns (ODPs) have been developed in order to provide standardized and re-usable solutions for such recurring problems in ontology building (Suárez-Figueroa & Gómez-Pérez, 2008). The use of ODPs should facilitate ontology design, ontology maintenance and ontology integration. In this section we focus on the use of ODPs as ontological resources in the development of ontologies.

The ODPs will be visualized by using images produced with the **U**nified **M**odeling **L**anguage (UML), which is a graphical modelling language for specification, design and documentation of systems.[1] The ODPs presented in this guideline can be classified into three main groups Aranguren (2005):

- **Extension ODPs** are ODPs that extend the limits of OWL 2. We will introduce the Exception pattern and the N-ary Relations pattern.
- **Good Practice ODPs** are used to produce more modular, efficient and maintainable ontologies. They address typical pitfalls of ontology engineering. We introduce the Normalisation pattern, the Closure pattern and the Value Partition pattern.
- **Content ODPs** are ODPs that are used to model a well-defined segment of a domain. They are therefore specific to this domain. We introduce the Process Sequence pattern and the Spatial Disjointness pattern.

6.2. Extension ODPs

6.2.1. Exceptions

Description The distinction of the normal (canonical) from the exceptional (non-canonical) is characteristic for biology and medicine. E.g., eukaryotic cells are commonly considered to be cells with a nucleus.[2] At the same time some biologists consider mammalian red blood cells as eukaryotic cells, although they lack a nucleus. If they are represented as a subclass of eukaryotic cells a logical error occurs.[3]

[1] http://www.omg.org/spec/UML/2.3/Infrastructure/PDF/, last access on 11.10.2012.

[2] An accurate classification of red blood cells is still possible by including the presence of S80 ribosomes in the definition.

[3] http://www.gong.manchester.ac.uk/odp/html/Exception.html, last access 09.07.2009.

Aim To model exceptions without breaking the strict class-subclass hierarchy and to get a consistent classification.

Application scenarios The exception ODP can be applied to a broad range of domains.

Structure/ Implementation For any exception class *A*,

- create two subclasses of *A*, one representing the *TypicalA* and one representing the *AtypicalA*
- add a covering axiom to *A* to state that instances of *A* are either typical or atypical, so that *A* is the disjoint union of *TypicalA* and *AtypicalA*
- the conditions that make *A* typical are pushed down into *TypicalA*
- all other subclasses of *A* are left unchanged

A covering axiom is added to the main class *EukaryoticCell* to state that all instances must belong to one or the other subclass *TypicalEukaryoticCell* or *AtypicalEukaryoticCell*. Hence, *EukaryoticCell* will be equivalent to the disjoint union of *TypicalEukaryoticCell* and *AtypicalEukaryoticCell*. After reasoning the correct hierarchy will be inferred with the two new subclasses 'Typical' and 'Atypical' at every level.

Warning The exception pattern may appear confusing because it requires auxiliary classes, which clutter up the class hierarchy. The exception ODP can produce ontologies that are too complex and difficult to manage.

Example A standard example for the Exception Pattern is the classification of cells:[4]

- All eukaryotic cells have one nucleus.
- Mammalian red blood cells are considered as eukaryotic cells, but they lack a nucleus. Thus they are a subclass of eukaryotic cells without fulfilling the condition for belonging to that class (having a nucleus).
- Avian red blood cells have a nucleus.

The important classes are the newly created classes: *TypicalEukaryoticCell*, *TypicalRedBloodCell*, *AtypicalEukaryoticCell* and *AtypicalRedBloodCell*. These classes will be inserted in any exception class (here *EukaryoticCell* and *RedBloodCell*). The rest of the classes in the hierarchy stays the same. This means that we make the following assertions:

AtypicalRedBloodCell `equivalentTo`
RedBloodCell `and` (**hasPhysicalPart** `some` *CellNucleus*)

RedBloodCell `subClassOf` *EukaryoticCell*

RedBloodCell `subClassOf` *TypicalRedBloodCell* `or` *AtypicalRedBloodCell*

[4] http://www.gong.manchester.ac.uk/odp/html/index.html, 09.07.2009

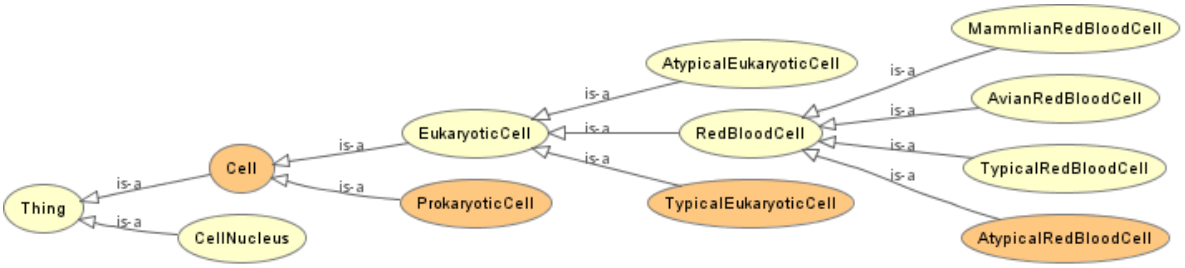

Figure 6.1.: Asserted Model: General view of typical and atypical subclasses before reasoning

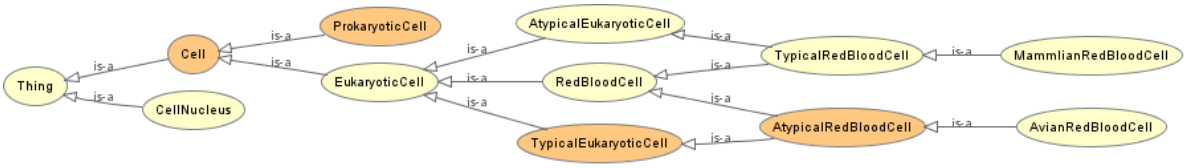

Figure 6.2.: Inferred Model: General view of typical and atypical subclasses after reasoning

TypicalRedBloodCell `subClassOf` *RedBloodCell*

AvianRedBloodCell `subClassOf` **hasPhysicalPart** `some` *CellNucleus*

AvianRedBloodCell `subClassOf` *RedBloodCell*

MammalianRedBloodCell `subClassOf` **hasPhysicalPart** `only`
`(not` *CellNucleus*`)`

MammalianRedBloodCell `subClassOf` *RedBloodCell*

6.2.2. N-ary Relations

Description For precise descriptions, relations with two arguments (A,B) as introduced are sometimes not sufficient. E.g., we want to relate two interacting molecules with the place where they interact, such as expressed by the relation **interacts** (*Substance*, *Substance*, *Place*). OWL, however does not support relations with more than two arguments.

Aim To find a way to express more than two-valued relations given the OWL restriction.

Application scenarios The N-ary Relations ODP shows how to alternatively represent a relation with n arguments.

Structure/ Implementation The solution is called reification. This means, that the meaning originally expressed as a relation is encoded in a class. Such classes are often thought of as processes, e.g. the relation **interacts** is expressed as the class *Interaction* under *Process*. Each original argument then corresponds to one class restriction for which standard relations such as **hasParticipant**, **hasAgent**, **hasLocus** can be used,

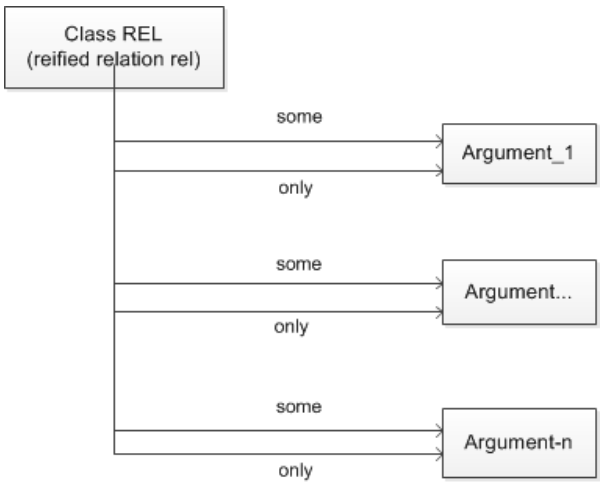

Figure 6.3.: N-ary Relationships

which has the advantage that the meaning is more explicit than if hidden within a list of arguments.

Example The above example **interacts** (*Substance*, *Substance*, *Place*), could then be modelled as

SubstanceInteraction `subClassOf` *Process* `and`
(**hasParticipant** `some` *Substance*) `and`
(**hasParticipant** `only` *Substance*) `and`
(**hasLocus** `some` *PhysicalEntity*)`and`
(**hasLocus** `only` *PhysicalEntity*)

This relation, however, does not specify the number of substances. This could be corrected by adding the restriction "**hasParticipant** `exactly 2`".

Note that we use a closure (cf. 6.3.2 on page 66), i.e. we require the existence of each argument and we constrain its value to a certain type.

Warning Reification introduces numerous problems into the ontology. For example, none of the usual OWL constructs for object properties/relations can be applied (i.e. it is not possible to specify transitivity/reflexivity etc. over reified relations). Additionally, uniqueness of the tuples that instantiate the relations can no longer be enforced, so that multiple distinct instances of the reified class connecting the same relata can be created.

6.3. Good Practice ODPs

6.3.1. Normalisation

Description The taxonomic backbone of ontologies is either monohierarchical or polyhierarchical. In monohierarchical (Figure 6.4) or strict hierarchies, every class can have

only one (immediate) superclass ("parent"), whereas polyhierarchies allow for multiple parents. Polyhierarchies are more difficult to create and to maintain due to their increasing structural complexity. Normalisation is an ontology building technique that relies on using an automated reasoner to maintain the polyhierarchy. The asserted taxonomy is therefore monohierarchical; all additional taxonomic links will be inferred by automatic reasoning from the class definitions. Hence there is no need to manually assert a polyhierarchy, as it can be inferred from an asserted monohierarchy. However, this requires fully defined classes [5].

Aim To produce polyhierarchies by maintaining only monohierarchies.

Application scenarios The 'Normalisation' ODP can be used in a multiple inheritance ontology to build an ontology with an efficient polyhierarchy structure.

Structure/ Implementation The following steps are to be followed Aranguren (2005)[6][7]:

1. Group all the classes you need in a tree form (strict hierachy)
2. Create primitive classes (only necessary conditions), that means classes with only one parent. (They are marked yellow in the example class hierarchy.)
3. Primitive siblings must be disjoint.
4. Redefine the classes and newly added classes according to the conditions for belonging to each class, that means define:
   - primitive classes (classes with necessary conditions) marked yellow and
   - defined classes (classes with necessary & sufficient conditions) marked orange

After reasoning you get a polyhierarchical structure (**D**irected **A**cyclic **G**raph, DAG).

Example According to the example from [8], the class '*Cell*' is a superclass of: '*CirculatingCell*', '*DefensiveCell*', '*AnimalCell*', '*PlantCell*', and '*StuffAccumulatingCell*'. '*AnimalCell*' has the following subclasses: '*Neutrophil*' and '*SyncytialGiantCell*'. '*PlantCell*' has the subclass '*MyrosinCell*'. The normalisation ontology has two parts:

1. The first part consists of primitive classes (in the figure yellow ovals), which have one superclass and are pair-wise disjoint.
2. The second part consists of defined classes (in the figure orange ovals), which have no superclasses, apart from owl:Thing (i.e. the root class) and are not disjoint.

After reasoning all entailed subsumptions are inferred, so that we get the following polyhierarchy structure .

The underlying axioms in Manchester notation:

[5] http://ontogenesis.knowledgeblog.org/49, 11.10.2012.
[6] http://ontogenesis.knowledgeblog.org/49, 11.10.2012
[7] http://www.gong.manchester.ac.uk/odp/html/index.html, 09.07.2009
[8] http://www.gong.manchester.ac.uk/odp/html/index.html, 09.07.2009.

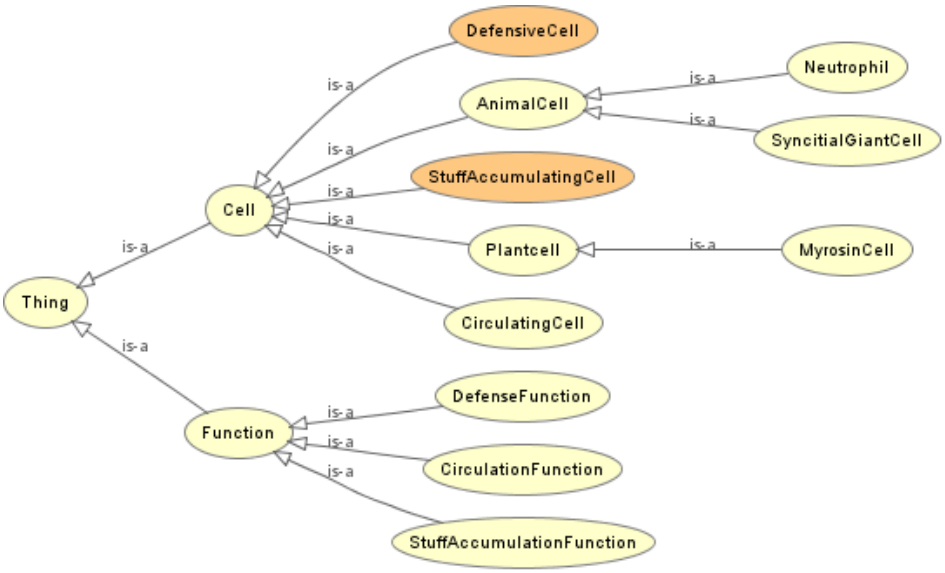

Figure 6.4.: Normalisation ODP in a strict hierachy: Asserted before reasoning

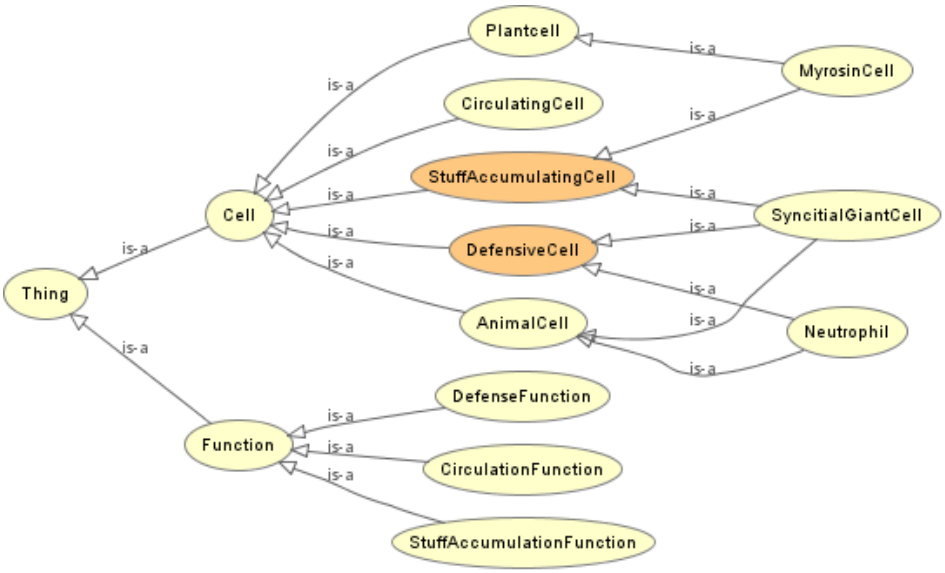

Figure 6.5.: Normalisation ODP in a Directed Acyclic Graph: Inferred after reasoning

Neutrophil `subclassOf` **bearerOf** `some` *CirculationFunction*

Neutrophil `subclassOf` **bearerOf** `some` *DefenseFunction*

MyrosinCell `subclassOf` **bearerOf** `some` *StuffAccumulationFunction*

DefensiveCell `equivalentTo` (*Cell* `and` **bearerOf** `some` *DefenseFunction*)

StuffAccumulation `equivalentTo`
(*Cell* `and` **bearerOf** `some` *StuffAccumulationFunction*)

6.3.2. Closure

Description The rationale of the Closure Pattern[9] is the open world assumption of OWL 2. This means that an existential restriction is not sufficient to 'close' a relationship. An additional value restriction is needed. For example, a carnivore eats meat and an herbivore eats vegetables and an omnivore eats food of both kinds. But if we only assert this, we do not exclude that a carnivore does *also* eat vegetables and that an herbivore does *also* eat meat. Thus we do have to assert explicitly that a carnivore eats meat and only meat, and that a herbivore eats vegetables and only vegetables. Without these value restrictions, *Carnivore* and *Herbivore* would appear as subclasses of *Omnivore*, whereas they are in fact disjoint classes.

Aim To express that something stands in a certain relation to some class and only that class.

Application scenarios The Closure Pattern is used to definine necessary and sufficient conditions on the same object property.

Structure/ Implementation Assert a statement with *existential restriction* and *value restriction* according to the following schema ("**relatedTo**" being a placeholder for an arbitrary object property):

A `subClassOf` (**relatedTo** `some` B) `and` (**relatedTo** `only` *B*)

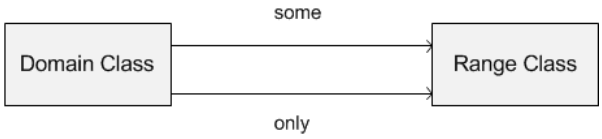

Figure 6.6.: Closure Ontology Design Pattern

Example The definition of the class *BacteriaPopulation* in BioTop usess the Closure Pattern:

[9] http://www.gong.manchester.ac.uk/odp/html/index.html, 09.07.2009.

BacteriaPopulation `equivalentTo` *Population* `and`
(**hasGranularPart** `some` *BacterialCell*) `and`
(**hasGranularPart** `only` *BacterialCell*)

This means that every instance of the class *BacteriaPopulation* has *some* instance of the class *BacterialCell* as its granular part, and that it has *only* instances of the class *BacterialCell*' as granular parts.

Warning Many new users confuse the use of "`some`" and "`only`". Be aware that value restrictions are sometimes also called "universal restrictions". Additionally, the characteristics of the object property over which closure is performed need to be taken into account. Especially closure over transitive properties can have unexpected results. For example, the closure statement "(**hasProperPhysicalPart** `only` *OxygenAtom*)" cannot be used in a definition of *OxygenMolecule* because it would forbid oxygen molecules from having neutrons, electrons or protons as parts, whereas oxygen atoms would be defined as having those parts, thus deriving a contradiction. The proper statement would thus be "`(`**hasProperPhysicalPart** `only (`*OxygenAtom* `or` *SubAtomicParticle*`))`".

6.3.3. Value Partitions

Description The Value Partition Pattern consists of a covering axiom and disjoint axioms which allow a precise description of the values a parameter may take. The features are constrained by having certain values. For example, a person can be short, medium or tall, but a person cannot have all these values of height at the same time. Similarly, the juridical gender of a person can only be male or female. The Value Partition ODP is used to model the fact that a parameter can only take certain values (Aranguren, 2009; Aranguren et al., 2008).[10]

Aim To model disjoint values of attributes.

Application scenarios The Value Partition Pattern can be used, if we have features that are constrained to have certain disjoint values.

Structure/ Implementation The Value Partition Pattern consists of a parameter class (e.g. *TaxonValueRegion*) and values of the parameter as subclasses of the parameter class (e.g. *KingdomFungiValueRegion*, *KingdomArchaeaValueRegion*, *KingdomBacteriaValueRegion*, ...). The following steps must be followed:

1. For each attribute create a class. In each attribute class create a subclass for every value and make them disjoint.
2. The attribute class will be fully defined by a covering axiom for the attribute, i.e. it will be defined to be equivalent to the union of classes value(1), value(2),..., value(n). This ensures that whenever a new class is added, it is added as a subclass of the values.

[10] Cf. also http://www.gong.manchester.ac.uk/odp/html/index.html, 09.07.2009.

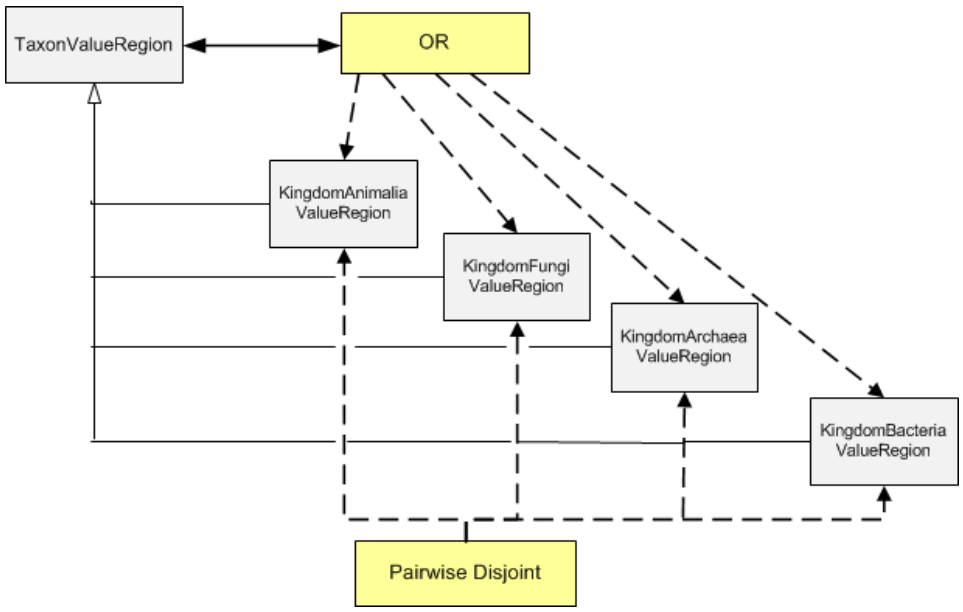

Figure 6.7.: The class '*TaxonValueRegion*'

Example The *TaxonValueRegion* from BioTop is a paradigmatic example of a value partition. The '*TaxonValueRegion*' has '*ValueRegion*' as superclass and the following subclasses: '*KingdomAnimaliaValueRegion*', '*KingdomFungiValueRegion*', '*KingdomArchaeaValueRegion*', '*KingdomBacteriaValueRegion*'. These subclasses or value classes are pairwise disjoint. The covering axiom is the following:

TaxonValueRegion `equivalentTo`
KingdomAnimaliaValueRegion `or`
KingdomFungiValueRegion `or`
KingdomArchaeaValueRegion `or`
KingdomBacteriaValueRegion

The attribute class is equivalent to the union of the value classes. Therefore if a new subclass is added to the attribute class, the reasoner will flag the attribute class to be inconsistent Aranguren (2009).

6.4. Content ODPs

6.4.1. Spatial disjointness

Description In BioTop, localization is expressed by the relation **'locusOf'** and its inverse **'hasLocus'**, which are transitive and relate a place with an entity which occurs, inheres, or is part of it.

Aim To specify that objects of the type A never overlap topologically with objects of type B.

Application scenarios It can be used in any ontology that describes the spatial composition of objects and places.

Structure/ Implementation For each class pair (A, B) we add similar restrictions in the following way:

A `subClassOf` **locusOf** `only` (not **hasLocus** `some` *B*)

B `subClassOf` **locusOf** `only` (not **hasLocus** `some` *A*)

Example It is recommended to carefully identify structural dissection levels in order to achieve completeness and parsinomy. For instance, if the following axioms obtain:

Lung `subClassOf` **partOf** `some` *Thorax*

Stomach `subClassOf` **partOf** `some` *Abdomen*

partOf `subPropertyOf` **hasLocus**

and if *Thorax* is "spatially disjoint" from *Abdomen*, expressed as

Abdomen `subClassOf` **locusOf** `only` (`not` **hasLocus** `some` *Thorax*)
Thorax `subClassOf` **locusOf** `only` (`not` **hasLocus** `some` *Abdomen*)

then the spatial disjointness of *Stomach* and *Lung* will be infered by the reasoner.

Warning The relations '**locusOf** ' and '**hasLocus**' may in some cases not be strict enough. For instance, the above axioms are debatable in case we want to model organisms that are located inside other organisms, e.g. a fetus inside its mother or a parasite within its host. In such cases, it might be more appropriate to use '**hasPart**' or '**partOf**' instead of '**locusOf**' or '**hasLocus**'.

A. Appendix: Using Protégé and its Reasoners

A.1. The Ontology Editor Protégé

There are several editors available to develop an ontology. These guidelines focus on the widely used editor Protégé 4.x which is freely available from http://protege.stanford.edu/doc/owl/getting-started.html. The download page offers different plattforms, like an installer program, ZIP file and OS X application bundle, between which the user can choose.. Useful information about installation, tutorials and other details are available under the following URLs:

- Protégé Websitehttp://protege.stanford.edu/
- Protégé Ontology Libraries
  http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library
- Protégé User Documentation
- http://protege.stanford.edu/doc/users.html
- Protégé tutorial
- http://www.co-ode.org/resources/tutorials/
- Plugins
  http://code.google.com/p/co-ode-owl-plugins/downloads/list?can=3

A.2. Tips and Tricks

In this section, we will present some of the technical methods that have been previously developed for using Protégé. It is important to show where the problems may lie and how to fix these problems. These technical methods may facilitate the use of Protégé.

A.2.1. Expand the Protégé Store (Memory)

The maximum amount of memory that a Java VM can use is 1.6 GB on Windows XP and 2 GB on most Unix machines. To set the heap size parameter, you must consider the following two points: (a) When setting the heap size parameter too low, this leads to the error message

"out of memory" (b) To set the heap size parameter too high, however, may cause the system to hang or lead to a weak performance [1].

To change the heap size start Protégé by double-clicking on the Protégé icon or using the start menu. To change the heap size parameter, you must update the *"Protege.lax"* file. By default, the *"Protege.lax"* file specifies a heap of 200MB for Protégé 4. To change the heap size in Protégé 4.x go to Protégé 4.x directory and open the file *"Protege.lax"*with any text editor. You find the following section in *"Protege.lax"*:

```
# LAX.NL.JAVA.OPTION.JAVA.HEAP.SIZE.MAX
# -------------------------------------
# maximum heap size
```

Put under this section the following line and set the heap size. The line is given in Protégé wiki page http://protegewiki.stanford.edu/wiki/Setting_Heap_Size:

lax.nl.java.option.java.heap.size.max=xxxxxx

If you have changed this line, save it and restart Protégé 4.x, because the change will be activated after restarting.

A.2.2. Protégé Update and Download New Plugins

Protégé 4.x can check automatically for updates and downloads for available new plugins. The automatic update is disabled in Protégé, you must choose the Protégé menu item File -> Preferences.

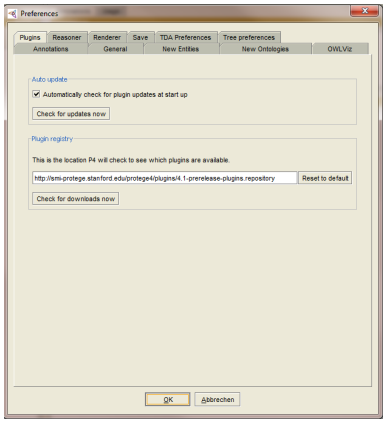

Figure A.1.: Plugins Preferences: Tab Plugins

Now the "Preferences" dialog appears. Choose the tab "Plugins" and activate the checkbox "Automatically check for plugin updates at start up". When you want to know which other new plugins are available to download, you can use the button "Check for downloads now". For update and downloads of new plugins, you need an internet connection. It will take several seconds for the dialog, to select which plugin you like to update or download, to appear. Updates will take effect the next time you start Protégé 4.x.

A.2.3. OWLViz

[1] http://protegewiki.stanford.edu/wiki/Setting_Heap_Size, 11.07.2011

OWLViz (Graphviz) is an open source graph visualization software. After the installation of Protégé 4.x you can download the GraphViz from http://www.graphviz.org/. For example, you can install GraphViz in *"C:\Program"*. After installing Graphviz, some configuration steps have to be executed.

To activate the OntoViz Tab in Protégé 4.x follow these steps:

- Please start Protégé 4.x by double-clicking on the Protégé icon
- Click in the menu "File" and than "Preferences". The following window (Figure 1.3) will appear.

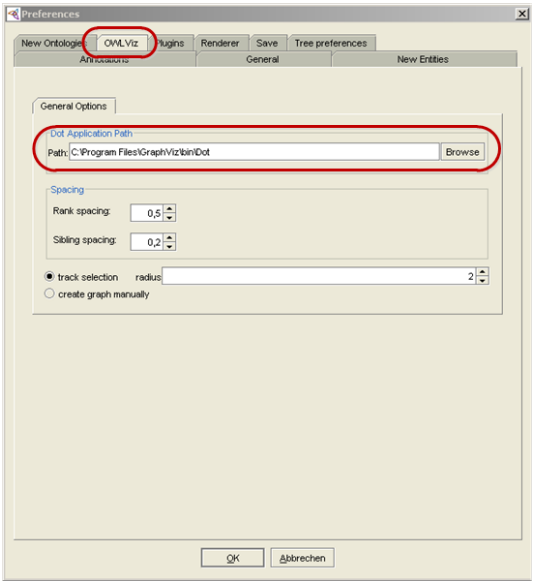

Figure A.2.: Preferences dialog: OWLViz Tab

Choose in Preferences the OWLViz tab. Then you must change the path to the location of the "DOT" executable (dot.exe) in "Dot Appplication Path". For example "C:\Program\GraphViz\bin\dot.exe", then click *"OK"*. You find "dot.exe" in file "GraphViz\bin".

After that, you must add the "GraphViz" tab in Protégé. Choose in the menu "Tabs" and activate the OWLViz. Now the "GraphViz" tab appears on the menu bar.

A.2.4. Proxy Setting in Protégé 4

Step 1: Please click on the directory where Protégé 4.x is installed e.g. "C:/Programs".

Step 2: Open Protégé 4.x directory.

Step 3: Click with the left mouse button on *"Protégé.lax"* and open it with "WordPad" or any other editor if you have.

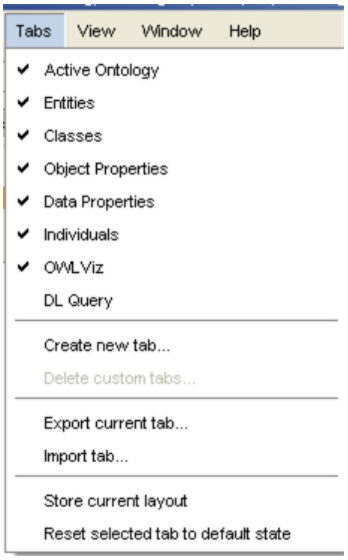

Figure A.3.: Protégé menu: In "Tabs" activate the OWLViz

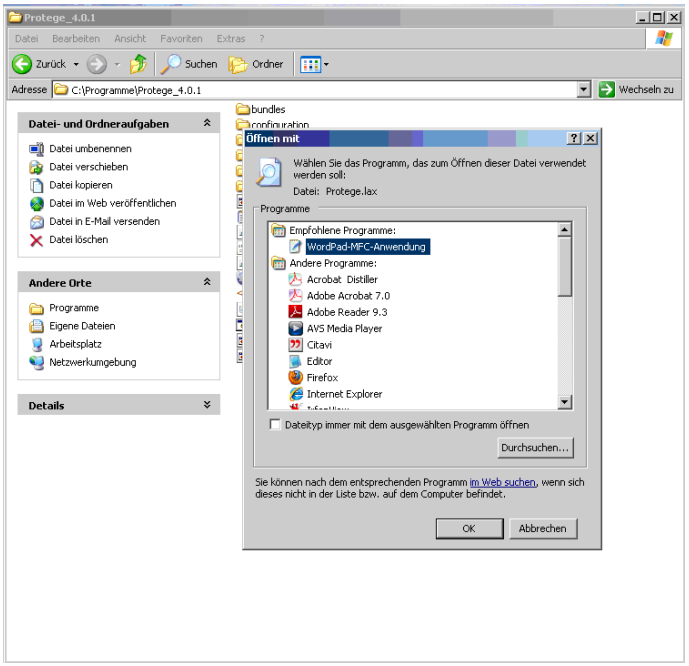

Figure A.4.: Opening the Protégé.lax

Step 4: Please scroll down the page until you see the section “Proxy connection”.

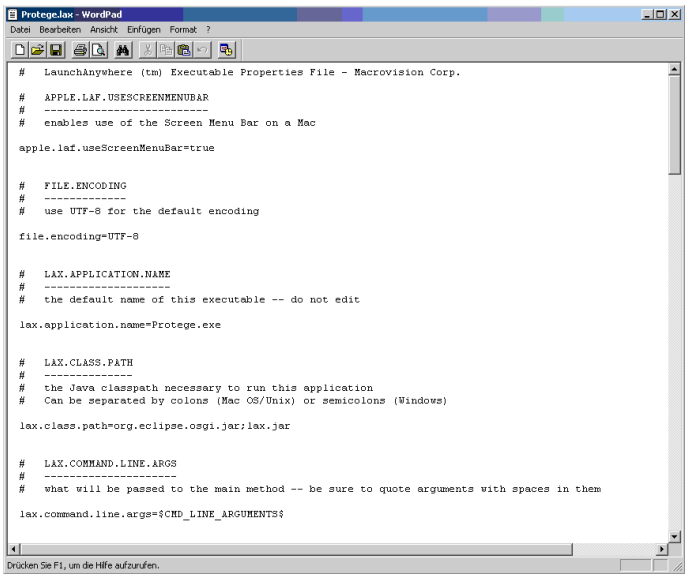

Figure A.5.: Protégé.lax file

Step 5: Enter the following lines in Protégé 4.x and save them:

```
# Proxy connection
# -----------------
# Add Proxy in Pro-
tege.lax in protege in-
stallation directory
"lax.nl.java.option.additional=-Dhttp.proxySet=true
-Dhttp.proxyHost=YOUR_PROXY_HOST
-Dhttp.proxyPort=YOUR_PROXY_PORT"
```

Note: This bit of code has to be entered in *Protégé.lax* as a single line (and not in multiple lines)!

To connect a proxy, you must add your*"Proxy_User"* and *"Proxy_Password"* in *"Protege.lax"* file like that:

```
# Proxy connection
# -----------------
# Add Proxy in Protégé.lax in Protégé installation directory
"lax.nl.java.option.additional=-Dhttp.proxySet=true
-Dhttp.proxyHost=YOUR_PROXY_HOST
-Dhttp.proxyPort=YOUR_PROXY_PORT
-Dhttp.proxyUser=YOUR_PROXY_USER
-Dhttp.proxyPassword=YOUR_PROXY_PASSWORD"
```

Note: Again, this bit of code has to be entered in *Protégé.lax* as a single line (and not in multiple lines)!

Proxy setting and authentication also descripted in "Search Mailing List Archives" by protege-discussion under this URL: https://mailman.stanford.edu/pipermail/protege-discussion/2007-February/000735.html

A.2.5. Import Biotop in Protégé 4.x

Step 1: Please open Protégé 4.x. Once you have opened Protégé you will come directly to this page. Now you can see the *"Ontology Imports"* tab.

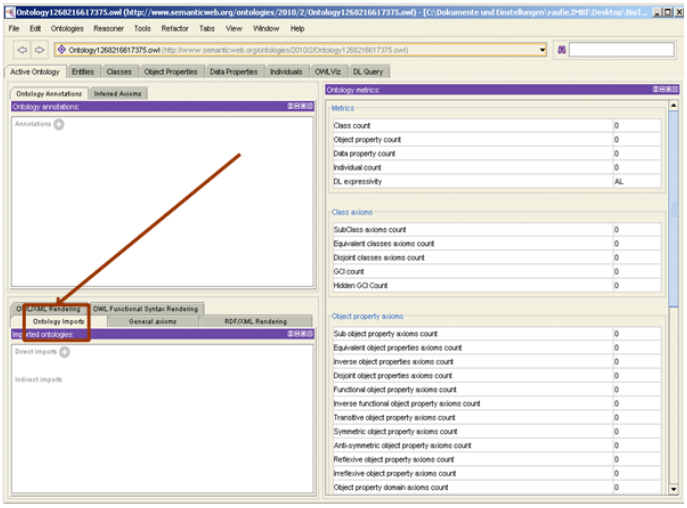

Figure A.6.: Ontology Imports Tab

Step 2: Now choose the “Direct imports”.

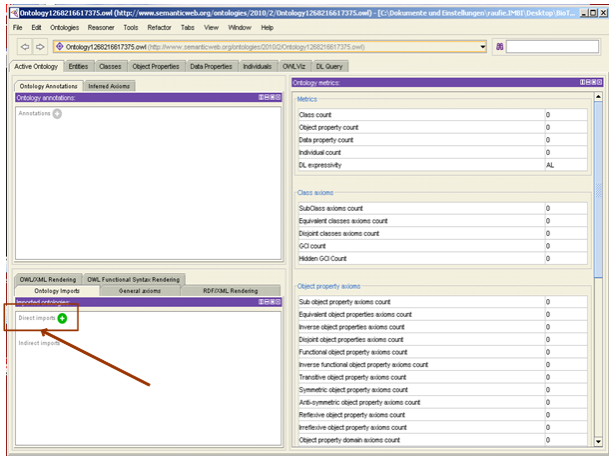

Figure A.7.: Direct imports

Step 3: Once you have opened Protégé 4.x, choose the radio button *“Import an ontology contained a document located on the web.”* and click the *“Continue”* tab.

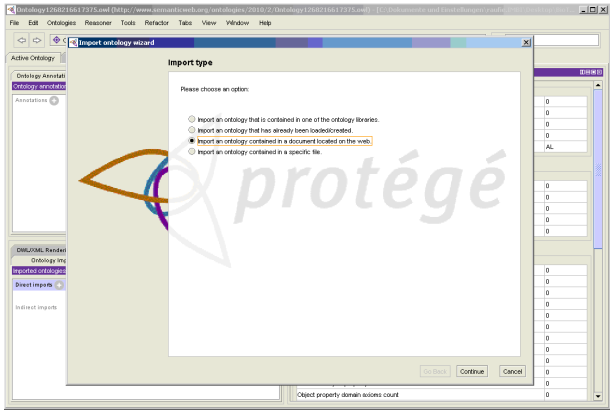

Figure A.8.: Import ontology wizard dialog: Several “Import type”

Step 4: Now click on the *“Bookmarked URIs”*.

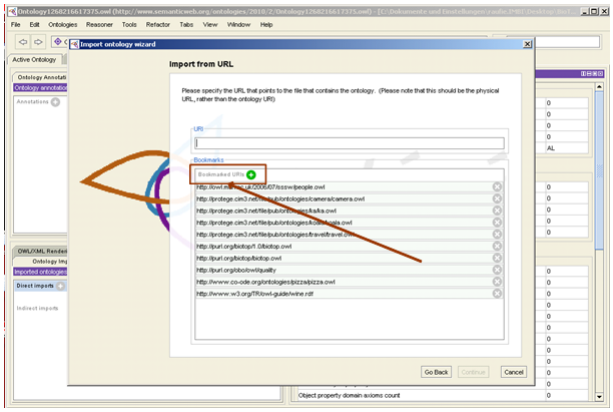

Figure A.9.: Import ontology wizard dialog: The “Bookmarked URIs”

Step 5: An URI window will open. Enter this URI: `http://purl.org/biotop/biotop.owl` and click *“OK”*.

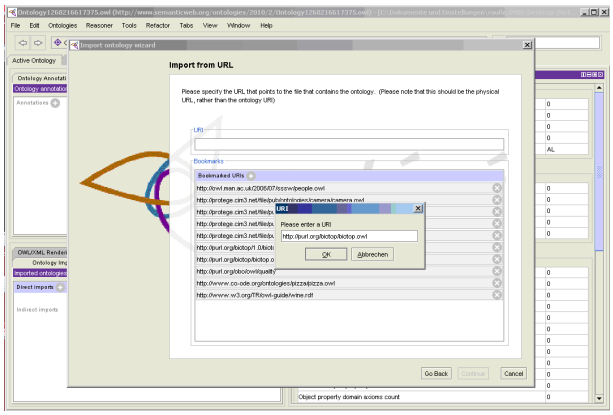

Figure A.10.: Import ontology wizard dialog: Import from URL

Step 6: Now, the URI is stored in *"Bookmarked URIs"*.

Step 7: Click on the URI `http://purl.org/biotop.biotop.owl` in the *"Bookmarked URIs"*.

The URI `http://purl.org/biotop.biotop.owl` will appear in *"URI"*, then click the *"Continue"* tab.

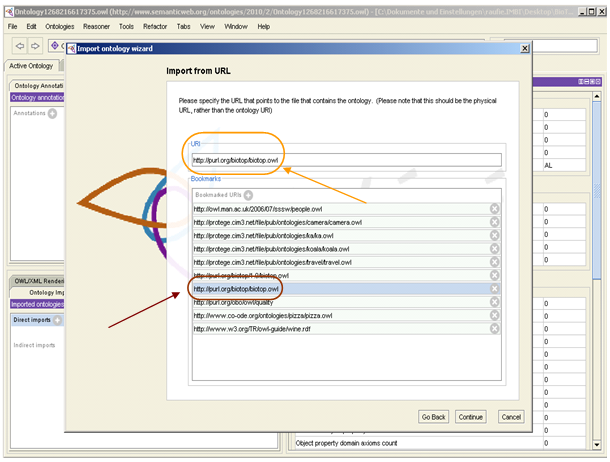

Figure A.11.: Import ontology wizard dialog: Choosing the ontology from Bookmarks

Step 8: Now *"biotop.owl"* is imported from the URI: `http://purl.org/biotop/biotop.owl`.

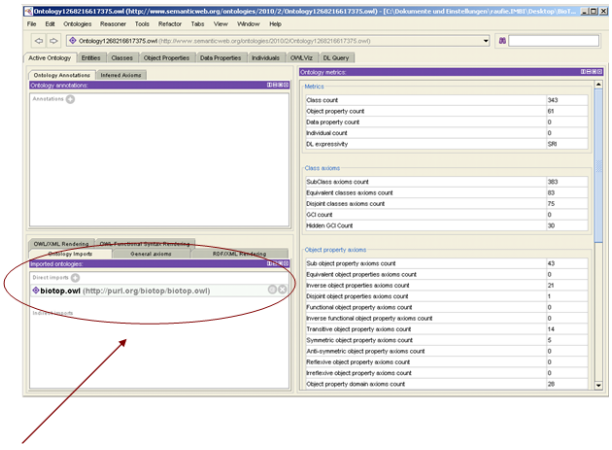

Figure A.12.: The selected ontology appears in "Direct Import"

A.3. Reasoner

A.3.1. What Is a Reasoner?

Ontologies are formal representations of statements about a domain in a machine-readable form. This allows the application of automated reasoning programs to these statements. Such

a reasoner takes all the axioms asserted in an ontology and infers new equivalences and subclass relations from these. That is, a reasoner makes explicit facts that previously have been only implicitly contained in the ontology. By doing this, the reasoner computes all classes that are explicitly or implicitly described in the ontology. This is called the *classification* of the ontology. This kind of reasoning is also necessary to process search queries within an ontology (cf. the DL Query tab in Protégé). During classification, the reasoner will also reveal inconsistencies in the ontology that otherwise would perhaps remain unnoted. Classes whose descriptions entail inconsistencies will be subsumed under a new class *Nothing* in the inferred ontology. Hence the three main tasks for reasoners are classification, query processing and consistency checking. It should be noted that for large and complex ontologies, classification can take a long time. Reasoning may lead to the following results:

- new subclass relations,
- propagation of properties from superclasses to subclasses,
- revelation of equivalences (marked by an equivalent sign „≡“ in the bullet before the class name),
- uncovering of inconsistencies („Nothing“).

A.3.2. Why Do We Use a Reasoner?

The reasoner is a useful tool both when we develop an ontology and before its application. During development, we use the reasoner to check the consistency of newly added classes. When this step is performed with each new class, certain modelling errors can immediately be detected and mitigated.

Ontologies are mostly used in their inferenced version, i.e. after a reasoning programme has made all inferences explicit. This allows, e.g., the 'inheritance' of properties from superclasses to subclasses. Moreover, it is a useful design strategy to assert only a mono-hierarchy, i.e. a hierarchy in which every item has at most one parent, because such hierarchies are easier to maintain. This does not mean that there are no cases of multiple inheritance at all, but only that there are no such cases explicitly asserted: All multiple inheritances are inferred by the reasoner. Also some design patterns like the exception pattern, 6.2.1, rely on the use of a reasoner in order to infer a poly-hierarchy from an asserted mono-hierarchy.

A.3.3. HermiT Reasoner

There are several reasoning programmes, and some of them are freely available. Among the reasoners that are pre-installed with Protégé, the reasoner HermiT can be recommanded for most purposes. HermiT can be started from the Reasoning menu. For doing this, HermiT has to be choosen in the menu before “Start Reasoner” is selected. The result can be view with the help of the tab “Class hierarchy (infered)”.

HerimiT can also be downloaded manually downloaded from http://hermit-reasoner.com/. After downloading, the file “org.semanticweb.HermiT.jar” has to be saved into the

plugin folder of Protégé. After the next start of Protégé, HermiT reasoner will appear in the Reasoner menu. Note that HermiT 1.2.x works with Protégé 4.1 alpha and HermiT 1.3.x works wih Protégé 4.1 beta.

List of Figures

List of Tables

Bibliography

Aranguren ME (2005). Ontology Design Patterns for the Formalisation of Biological Ontologies. M. Phil. thesis, University of Manchester, Manchester. URL http://org.buffalo.edu/RTU/papers/assisted/citations/MPhilThesis.pdf.

Aranguren ME (2009). Role and Application of Ontology Design Patterns in Bioontologies. Ph. D. thesis, University of Manchester, Manchester. URL http://mikeleganaaranguren.files.wordpress.com/2010/01/thesis.pdf.

Aranguren ME, Antezana E, Kuiper M & Stevens R (2008). Ontology Design Patterns for bio-ontologies: a case study on the Cell Cycle Ontology. *BMC Bioinformatics*, 9(Suppl 5):S1. ISSN 1471-2105. doi:10.1186/1471-2105-9-S5-S1.

Armstrong DM (1980). *A Theory of Universals*, *Universals and scientific realism*, vol. 2. Cambridge University Press, Cambridge, 1st edn. ISBN 9780521280327.

Baader F, Calvanese D, McGuinness D, Nardi D & Patel-Schneider P (2010). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, Cambridge, 2nd edn. ISBN 9780521781763.

Beisswanger E, Schulz S, Stenzhorn H & Hahn U (2008). Biotop: An Upper Domain Ontology for the Life Sciences: A Description of its Current Structure, Contents, and Interfaces to OBO Ontologies. *Applied Ontology*, 3(4):205–212. doi:10.3233/AO-2008-0057.

Boeker M, Schober D, Raufie D, Grewe N, Röhl J, Jansen L & Schulz S (2012). Teaching Good Biomedical Ontology Design. In Cornet R & Stevens R (eds.), *Proceedings of the 3rd International Conference on Biomedical Ontology (ICBO 2012), KR-MED Series, July 21-25, 2012, Graz, Austria*. URL http://ceur-ws.org/Vol-897/sessionJ-paper25.pdf.

Bozsak E, Ehrig M, Handschuh S, Hotho A, Maedche A, Motik B, Oberle D, Schmitz C, Staab S, Stojanovic L, Stojanovic N, Studer R, Stumme G, Sure Y, Tane J, Volz R & Zacharias V (2002). KAON — Towards a large scale Semantic Web. In Bauknecht K, Tjoa A & Quirchmayr G (eds.), *E-Commerce and Web Technologies*, *Lecture Notes in Computer Science*, vol. 2455, pp. 231–248. Springer, Berlin. ISBN 978-3-540-44137-3.

Gangemi A, Guarino N, Masolo C, Oltramari A & Schneider L (2002). Sweetening Ontologies with DOLCE. In Gómez-Pérez A & Benjamins R (eds.), *Knowledge Engineering and Knowledge Management*, pp. 166–181. Springer, Berlin. ISBN 9783540442684. doi:10.1007/3-540-45810-7_18.

Grenon P (2003). BFO in a Nutshell: A Bi-categorial Axiomatization of BFO and Comparison with DOLCE: IFOMIS REPORTS 06/2003. URL http://www.ifomis.org/Research/IFOMISReports/IFOMIS%20Report%2006_2003.pdf.

Grenon P & Smith B (2004). SNAP and SPAN: Towards Dynamic Spatial Ontology. *Spatial Cognition & Computation*, 4(1):69–104. ISSN 1387-5868. doi:10.1207/s15427633scc0401_5.

Grenon P, Smith B & Goldberg L (2004). Biodynamic Ontology: Applying BFO in the Biomedical Domain. *Studies in Health Technology and Informatics*, 102:20–38. ISSN 0926-9630.

Gruber T (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220. ISSN 10428143. doi:10.1006/knac.1993.1008.

Guarino N & Welty C (2009). An Overview of OntoClean. In Staab S & Studer R (eds.), *Handbook on Ontologies*, pp. 201–220. Springer, Berlin. ISBN 978-3-540-70999-2.

Hekkala E, Shirley MH, Amato G, Austin JD, Charter S, Thorbjarnarson J, Vliet KA, Houck ML, Desalle R & Blum MJ (2011). An ancient icon reveals new mysteries: mummy DNA resurrects a cryptic species within the Nile crocodile. *Molecular Ecology*, 20(20):4199–4215. ISSN 1365-294X. doi:10.1111/j.1365-294X.2011.05245.x.

Hofweber T (2012). Logic and Ontology. In Zalta EN (ed.), *The Stanford Encyclopedia of Philosophy*. Stanford. ISBN ISSN 1095-5054. URL http://plato.stanford.edu/entries/logic-ontology/#4.1.

Jansen L (2006). Aristoteles' Kategorie des Relativen zwischen Dialektik und Ontologie. In Meixner U & Newen A (eds.), *Philosophiegeschichte und logische Analyse /Logical Analysis and History of Philosophy / History of ontology and a focus on Plato /Geschichte der Ontologie und ein Schwerpunkt zu Platon: 9*, pp. 79–104. Mentis. ISBN 389785158X.

Jansen L (2008a). Categories: The Top-Level Ontology. In Munn K & Smith B (eds.), *Applied Ontology: An Introduction*, pp. 173–196. Ontos Verlag, Heusenstamm bei Frankfurt. ISBN 9783938793985.

Jansen L (2008b). Classification. In Munn K & Smith B (eds.), *Applied ontology: an introduction*, vol. 9, pp. 159–172. Ontos Verlag.

Jansen L (2010). What is a Formal Ontology? Some Meta-Ontological Remarks. In Mainzer K (ed.), *ECAP10. VIII European Conference on Computing and Philosophy*, pp. 256–260. Hut, München.

Jansen L & Schulz S (2011). Grains, Components and Mixtures in Biomedical Ontologies. *Journal of Biomedical Semantics*, 2(Suppl 4):S2.

Johnson W (1921). *Logic*, vol. 1. Cambridge University Press, Cambridge.

Klyne G & Carroll JJ (2004). Resource Description Framework (RDF): Concepts and Abstract Syntax. URL http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/.

Masolo C, Borgo S, Gangemi A, Guarino N, Oltramari A & Schneider L (2003). WonderWeb Deliverable D17: The WonderWeb Library of Foundational Ontologies: Preliminary Report. ISTC-CNR, Padova. URL http://wonderweb.semanticweb.org/deliverables/documents/D17.pdf.

Merrill GH (2010). Ontological Realism: Methodology or Misdirection? *Applied Ontology*, 5(2):79–108. doi:10.3233/AO-2010-0076.

Motik B, Patel-Schneider PF & Cuenca Grau B (2009). OWL 2 Web Ontology Language Direct Semantics. URL http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/.

Munn K & Smith B (eds.) (2008). *Applied Ontology: An Introduction*. Ontos Verlag, Heusenstamm bei Frankfurt. ISBN 9783938793985.

Quine WV (1948). On What There Is. *The Review of Metaphysics*, 2(5):21–38.

Rector A (2008). Barriers, approaches and research priorities for integrating biomedical ontologies: SemanticHEALTH Deliverable D 6.1: Semantic Interoperability Deployment and Research Roadmap. URL http://www.semantichealth.org/DELIVERABLES/SemanticHEALTH_D6_1.pdf.

Röhl J & Jansen L (2011). Representing dispositions. In Herre H, Hoehndorf R, Kelso J & Schulz S (eds.), *Proceedings of Ontologies in Biomedicine and Life Sciences (OBML 2010), Mannheim, Germany*, vol. 2, p. S4. Journal of Biomedical Semantics, Mannheim and Germany. doi:10.1186/2041-1480-2-S4-S4. URL http://www.jbiomedsem.com/content/2/S4/S4/comments.

Scheuermann RH, Ceusters W & Smith B (2009). Toward an Ontological Treatment of Disease and Diagnosis. *Proceedings of the 2009 AMIA Summit on Translational Bioinformatics*, 2009:116–120. ISSN 2153-6430.

Schulz S & Hahn U (2007). Towards the ontological foundations of symbolic biological theories. *Artificial Intelligence in Medicine*, 39(3):237–250. doi:10.1016/j.artmed.2006.12.001.

Schulz S, Spackman K, James A, Cocos C & Boeker M (2011). Scalable representations of diseases in biomedical ontologies. *Journal of Biomedical Semantics*, 2(Suppl 2):S6. ISSN 2041-1480. doi:10.1186/2041-1480-2-S2-S6.

Schulz S, Stenzhorn H & Boeker M (2008). The ontology of biological taxa. *Bioinformatics*, 24(13):i313–321. ISSN 1367-4811.

Schulz S, Stenzhorn H, Boeker M & Smith B (2009). Strengths and limitations of formal ontologies in the biomedical domain. *RECIIS - Electronic Journal in Communication, Information and Innovation in Health*, 3(1):31–45. ISSN 1981-6286. doi: 10.3395/reciis.v3i1.241en.

Schwarz U & Smith B (2008). Ontologische Relationen. In Jansen L & Smith B (eds.), *Biomedizinische Ontologie*, pp. 155–172. VDF Hochschulverlag AG, Zürich. ISBN 3728131830.

Smith B (2004). Beyond Concepts: Ontology as Reality Representation. In Varzi A & Vieu L (eds.), *Proceedings of FOIS 2004. International Conference on Formal Ontology and Information Systems*, pp. 73–84.

Smith B (2005). Against Fantology. In Marek JC & Reicher ME (eds.), *Experience and Analysis*, pp. 135–170. ÖBV & HPT, Vienna.

Smith B & Ceusters W (2007). Ontology as the Core Discipline of Biomedical Informatics: Legacies of the Past and Recommendations for the Future Direction of Research. In Dodig-Crnkovic G & Stuart S (eds.), *Computing, Philosophy, And Cognitive Science - The Nexus and the Liminal*, pp. 104–122. Cambridge Scholars Press, Cambridge.

Smith B, Ceusters W, Klagges B, Köhler J, Kumar A, Lomax J, Mungall C, Neuhaus F, Rector AL & Rosse C (2005). Relations in biomedical ontologies. *Genome Biology*, 6(5):R46. ISSN 1465-6914. doi:10.1186/gb-2005-6-5-r46. URL http://genomebiology.com/2005/6/5/R46.

Smith B, Kusnierczyk W, Schober D & Ceusters W (2006). Towards a Reference Terminology for Ontology Research and Development in the Biomedical Domain. In Bodenreider O (ed.), *KR-MED 2006 Proceedings. Second International Workshop on Formal Biomedical Knowledge Representation*, pp. 57–65. Baltimore.

Spear AD (2006). Ontology for the Twenty First Century: An Introduction with Recommendations. URL http://www.ifomis.org/bfo/documents/manual.pdf.

Stenzhorn H, Beisswanger E & Schulz S (2007). Towards a Top-Domain Ontology for Linking Biomedical Ontologies. *Studies in Health Technology and Informatics*, 129(Pt 2):1225–1229. ISSN 0926-9630.

Studer R, Benjamins V & Fensel D (1998). Knowledge Engineering: Principles and methods. *Data & Knowledge Engineering*, 25(1–2):161–197. ISSN 0169-023X. doi:10.1016/S0169-023X(97)00056-6.

Suárez-Figueroa MC & Gómez-Pérez A (2008). First Attempt towards a Standard Glossary of Ontology Engineering Terminology. In Nistrup Madsen B (ed.), *Managing Ontologies and Lexical Resources: TKE 2008 : 8th International Conference on Terminology and Knowledge Engineering*. Litera, Græsted. ISBN 87-91242-50-9.