2 Scientific Background

2.1 Ontology in Philosophy

Originally, Ontology is a philosophical discipline concerned with the question of what exists and what is the essence of things. The term 'Ontology' stems from ancient Greek and can be translated as 'theory of existence'[4]. The discipline of Ontology was founded by Greek philosophers, such as Parmenides of Elea and Aristotle, during the 4th Century BC. Ontology has been a topic of active research throughout the Middle Ages and Modern Age until today, with contributions from such renowned philosophers as Kant and Wittgenstein. Nowadays, Ontology constitutes an important area of contemporary philosophy, covering large research projects and reaching out to such different areas as artificial intelligence, database theory, and natural language processing.
According to the Stanford Encyclopedia of Philosophy (cf. Hofweber 2005), the discipline of modern Ontology comprises four different aspects, denoted by (O1) to (O4):

(O1) The study of what there is, what exists.

(O2) The study of the most general features and relations of the entities which do exist.

A prerequisite for (O1) is to clarify in which things one must (initially) believe before one may reason about the existence of other things. Therefore, Ontology also includes

(O3) the study of ontological commitment, i.e., to become aware of what one is committed to.

Generally, an ontological commitment to the existence of an entity (A) becomes necessary in order to make a statement about the existence of another entity (B). In other words: the existence of entity A is presupposed or implied by asserting the existence of entity B. A typical commitment would be the choice of a modeling language (cf. Sect. 2.3); that is, one commits to abstract entities, such as classes or relations, or to particular theories, such as second order logic
Finally, the field of Ontology incorporates

(O4) the study of Meta-Ontology, i.e., saying what task it is that the discipline of Ontology should aim to accomplish, if any, how the questions it aims

[4] ὄ (Ontos), the genitive of ὄ (On), means '*of being*'; the suffix - (-logia) denotes a *science*, *study*, or *theory*. So originally, the word signifies '*theory of being*'.

W. Marquardt et al., *OntoCAPE*, RWTHedition, DOI 10.1007/978-3-642-04655-1_2,

to answer should be understood, and with what methodology they can be answered.

In the following, a particular ontological theory is referred to as an *ontology*[5]. The individual ontologies considered in the context of this work will mainly focus on the aspects (O1) and (O2).
An ontology can be specified on different levels of formality. According to Uschold and Grüninger (1996) and Hofweber (2005), an ontology is designated as

- informal if expressed in natural language;
- *semi-informal* if expressed in a restricted and structured form of natural language;
- *semi-formal* if expressed in an artificial and formally defined language; and
- *(rigorously) formal* if the ontology contains precise mathematical definitions of certain entities in terms of their properties and their relations to other entities. Such definitions are usually given in form of axioms formulated in a logic-based language. This allows proving certain properties about an ontology, such as its consistency[6].

Formal ontologies have proven to be applicable in numerous areas; a particularly popular field of application is based on utilizing a formal ontology as a framework for *information representation*.. Information represented in such a framework is easily accessible to automated information processing. For that reason, ontologies have become a subject of intensive research in the area of computer science.

2.2 Ontology in Computer Science

Over the last decades, the term ‘ontology’ has been adopted by computer scientists, firstly in the field of artificial intelligence (AI) and more recently in other areas, as well. Within this community, the term is used in a more narrow sense than in the context of philosophy, denoting a *formal ontology for information representation*[7] (see above). Viewed from the perspective of an AI system, this conception of an ontology is equivalent to the original philosophical definition of Ontology as a “theory of existence”, since, as Gruber (1995) put it, “for AI systems, what ‘exists’ is that which can be [formally] represented”.

---

[5] Adopting a proposal of Guarino and Giaretta (1995), we use the uncountable noun ‘Ontology’ (with capital ‘O’) to refer to the philosophical discipline; in contrast, the countable noun ‘ontology’ (with lowercase ‘o’) refers to a specific ontological theory, such as ‘Aristotle’s ontology’ or ‘the Cyc ontology’.

[6] An ontology is said to be consistent if it does not contain any logically conflicting statements.

[7] ‘Knowledge representation’ is often used synonymously with ‘information representation’.

In computer science, there are two principal types of usage for an ontology:

- The first type of usage has already been explicated in Chap. 1: An ontology serves as a library of knowledge components to efficiently build intelligent systems. To this aim, the generic ontology is to be transformed (i.e., extended and customized) into a knowledge base according to the requirements of the respective application.
- The second type of usage is as a *shared vocabulary* for communication between interacting human and/or software agents. According to their respective functions, the communicating agents may have different knowledge bases, but all the knowledge bases must be consistent with the ontology (Gruber 1995).

Both types of usage make the same demand on the ontology: They both require a consensual knowledge representation that is reusable in different application contexts. For the first case, this is obvious and has been extensively discussed in Chap. 1. As for the second case, the communicating agents perform different tasks requiring different knowledge bases, and thus the ontology must be suitable for each of these[8]. Thus, a properly crafted ontology should be applicable to both types of usage. As will be explained in Chaps. 11 and 12 OntoCAPE originally started as a shared vocabulary, but later evolved to a library for building knowledge-based systems.

Guarino (1998) points out that in philosophy, the term 'ontology' denotes a conceptual framework, whereas in computer science, 'ontology' often (but not always) refers to the engineering artifact used to represent such a conceptual framework:

> "In the philosophical sense, we may refer to an ontology as a particular system of categories accounting for a certain vision of the world. As such, this system does not depend on a particular language: Aristotle's ontology is always the same, independently of the language used to describe it.
> On the other hand, in its most prevalent use in AI, an ontology refers to an engineering artifact, constituted by a specific vocabulary used to describe a certain reality, plus a set of explicit assumptions regarding the intended meaning of the vocabulary words."

In this book, the term 'ontology' is used ambiguously with both meanings. If necessary, we will use the phrases 'ontology at the syntactic level' to refer to the engineering artifact, and 'ontology at the semantic level' to refer to the abstract conceptual framework[9].

---

[8] The only difference is that, in the first case, the ontology is directly reused for building the knowledge base, whereas this is not necessarily true in the second case. Yet even if a knowledge base has not developed directly from the ontology, it must still be consistent with the ontology's definitions.

[9] The synonymous terms 'symbolic level' and 'knowledge level', first suggested by Newell (1982), are also used in the literature.

Note that in computer science, the term 'ontological commitment' has a special meaning, as well: If some human or software agents agree on using an ontology for a given task in a consistent manner, they are said to commit to that ontology (Gruber and Olsen 1994; Studer et al. 1998). In other words, "an agent commits to an ontology if its observable actions are consistent with the definitions in the ontology" (Gruber 1995).

2.3 Representation of Formal Ontologies

Ontologies can be modeled with different modeling techniques, and they can be implemented in various kinds of languages (Uschold and Grüninger 1996). Examples of common modeling techniques, or *modeling paradigms*, include frames (e.g., Minsky 1975), first-order logic (e.g., Hodges 1983), description logic (abbr. DL; e.g., Baader et al. 2003), database modeling techniques (e.g., Chen 1976), and rule-based languages (a.k.a. rule languages; e.g., Lloyd 1987); for each paradigm, multiple implementations, or *modeling languages*, exist.
In spite of their diversity, the different modeling languages share structural similarities and have comparable modeling elements. In particular, most languages provide constructs for *classes*, *individuals*, *relations*, and *attributes*, although they may be named differently in the respective implementations. Moreover, some language allow for the definition of *axioms*. In the following, these different model components will be described in detail.

2.3.1 Classes and Individuals

A *class* represents a collection of entities that share a common characteristic. Depending on the respective modeling paradigm, classes are also denoted as *concepts* or *frames*. If referred to in the text, class identifiers are highlighted by *italicized sans-serif font*.
Entities that belong to a particular class are said to be *instances* or *members* of that class; for example, **water** and **ethanol** are instances of the *substance* class. Some modeling languages allow for the definition of *metaclasses*, the instances of which are again classes. The instances of an ordinary class are called *individuals*. Throughout this book, individuals are accentuated by **bold sans-serif font**.
Classes can be hierarchically organized by means of *subsumption relations*, which are also known as *specialization relations* or *subclassing relations*: The class *B* is said to be a specialization or a subclass of the class *A* if every instance of *B* is also an instance of *A*. In this case, *B* is said to be *subsumed* by *A*, and *A* is called a *superclass* of *B*.

By means of *axioms* (see below), it is possible to state certain properties about a class, such as the existence of *relations* (see below). In this context, two types of classes can be distinguished:

- *Primitive classes* have only necessary conditions (expressed in terms of their properties) for membership: An instance of a primitive class must always comply with the properties of that class, but there may be other individuals with the same properties which are not members of that class. Consequently, membership to a primitive class must be explicitly stated.
- *Defined classes* are characterized by necessary and sufficient conditions for membership. Thus, an individual whose properties match those of a defined class is automatically inferred to be a member of that class. Similarly, the subclasses of a defined class can be inferred if their properties match the class definition.

Most languages support *inheritance* between the classes in a subsumption hierarchy; that is, a subclass inherits all the properties of its superclass. Some languages allow for *multiple inheritance*, which means that a particular class can inherit properties from more than one superclass.

2.3.2 Relations

A *relation* represents an interrelation between some classes; depending on the respective modeling paradigm, relations are also called *properties*, *roles*, *slots*, or *associations*. While most modeling languages only provide modeling constructs for *binary relations* (i.e., relations between exactly two classes), a few have built-in constructs for *higher-arity relations* (a.k.a. *n-ary relations*) involving three or more classes. In the following, the term 'relation' is synonymously used for 'binary relation'. Relation identifiers will be denoted by sans-serif font throughout the text.

By default, a relation is (uni-)directional, which means that it points from a particular *domain* class to a designated *range* class: As an example, consider the relation hasReactant, which refers from a *chemical reaction* (its domain) to a *substance* (its range).

A relation can be instantiated, which means that it can be applied between an instance of the domain class and an instance of the range class. For example the above hasReactant relation can refer from the **esterification of acetic acid** (an instance of *chemical reaction*) to the individual **ethanol**. Unlike a class instance, an instantiated relation is not given a specific name but is identified via its domain and range individuals.

Some languages allow to further specify the relations by means of *relation properties* (sometimes called *property characteristics*). The following relation properties

are quite common, although a single language does not necessarily support all of them:

- A relation may be associated with another relation denoting its *inverse* – for example, isReactantOf would be the inverse of hasReactant, thus pointing from a *substance* to a *chemical reaction*.
- Alternatively, a relation may be declared to be *symmetric* – in this case, it is equivalent to its own inverse: A concrete example is the isEqualTo relation – it implies that, if A is equal to B, then B is equal to A, as well.
- A different property is *antisymmetry*, which is defined as follows: Given an antisymmetric relation R and two entities, A and B. If A is R-related to B, and B is R-related to A, then A and B must be identical. Note that symmetry and antisymmetry are not mutually exclusive – for instance, the isEqualTo relation is both symmetric and antisymmetric.
- Additionally, a relation may be declared to be *transitive*. This means that if entities A and B are related via a transitive relation R, and so are B and C, then A and C must also be R-related. A concrete example would again be the isEqualTo relation – if A equals B, and if B equals C, then A equals C.
- A relation may be declared to be *reflexive*, meaning that each entity to which a reflexive relation R is applicable is R-related to itself. For instance, the isEqualTo relation is reflexive since each entity is equal to itself.
- Alternatively, a relation R may be declared to be *irreflexive*; in consequence, an entity can never be R-related to itself. The relation isGreaterThan is a typical example of an irreflexive relation.
- A *functional* relation (sometimes also referred to as a *function*) cannot have more than one unique range individual; if a domain individual is related to more than one range individual via a functional relation, it will be concluded that the range individuals are identical. For obvious reasons, this property should not be combined with transitivity.
- The opposite effect is caused by an *inverse-functional* relation: If two domain individuals are related to the same range individual via an inverse-functional relation, it will be inferred that the domain individuals are identical. Thus, the range individuals of an inverse-functional relation can be utilized as unique identifiers for the domain individuals. Note that the inverse of a functional relation is automatically an inverse-functional relation.

A few modeling languages treat subsumption as a special case of a (transitive, reflexive, and antisymmetric) relation. Other languages allow for a hierarchical organization of relations, which is similar to that of classes. Unlike in class hierarchies, a *subrelation* may have properties different from those of its *superrelation*.

2.3.3 Attributes

Attributes represent features, characteristics, or parameters of classes and their instances. An attribute is identified by its name; it takes one or several values, which are specific to the class or instance the attribute is attached to. Usually, the values of a particular attribute are restricted to a specific datatype such as boolean, string, or integer.
Often, the same modeling constructs are used for the representation of relations and attributes; they differ from each other only with respect to their ranges: The range of a relation is given by its range class, whereas the range of an attribute is specified by its datatype. Due to the absence of a range class, most of the above relation properties cannot be applied to attributes. However, it is possible to declare an attribute to be functional or inverse functional; also, attributes may be hierarchically ordered.

2.3.4 Axioms

An *axiom* models a proposition or sentence that is always true. Generally, axioms provide an additional means for knowledge representation: They allow formalizing such knowledge that goes beyond stating the mere existence of classes, relations, and instances. Therefore, modeling paradigms that include axioms have a greater expressiveness than those without. In particular, axioms serve

- to explicitly define the semantics (or at least to constrain the possible interpretations and uses) of an ontological concept by imposing constraints on its values and/or its interactions with other concepts in the ontology;
- to verify the consistency of the knowledge represented in the ontology; and
- to infer new (i.e., formerly implicit) knowledge from the explicitly stated facts.

Formal axioms may be embedded in class or relation definitions, where they specify the properties of the respective class or relation. In fact, the declaration of the above introduced relation properties is usually realized by means of embedded axioms.
The following are common types of class-embedded axioms, stating

- the *disjointness* of classes – if the classes A and B are declared to be disjoint, then an instance of class A cannot simultaneously be an instance of class B;
- the *equivalence* of classes, meaning that such classes have precisely the same instances;
- the *extension* of a class by means of an explicit *enumeration* of its members.

Another common type of class-embedded axioms puts constraints on the relations originating from the respective class. Unlike relation properties, which are universally valid, these constraints are specific to the domain class, i.e., they are only locally valid. These *local constraints* include, but are not restricted to

- *(local) range restrictions*, stating that the range of a relation originating from the domain class is restricted to certain classes;[10]
- *cardinality constraints*, which specify either the exact number or the maximum/minimum number of range individuals for a given relation;
- *qualified cardinality restrictions* (a.k.a. qualified cardinality constraints, abbr.: QCR), which, in addition to specifying the number of range individuals, also prescribe the range class of which the individuals are to be instantiated from.[11]

The above introduced basic axiom types can be combined to more complex expressions. To this end, ontology languages provide additional constructors, such as the set operators of *union*, *intersection*, and *complement*.
Finally, *rules* constitute a further, very powerful mechanism for stating axioms. A rule axiom consists of an *antecedent* (or *rule body*) and a *consequent* (or *rule head*). Both the antecedent and the consequent are logical expressions, which are formulated in terms of the other constructs of the modeling language. Whenever the expression specified in the antecedent holds true, then the expression specified in the consequent must also hold. Thus, if an antecedent matches the current state of the ontology, then the consequent is affirmed, i.e., added to the ontology. Note that, while the antecedent is not necessarily true, the rule as a whole is universally valid, and therefore matches the above definition of an axiom; 'classical' axioms (i.e., axioms without a precondition) can be modeled as rules with an empty rule body.

2.3.5 Modularization

Virtually all of the modern ontology modeling languages support the modularization of ontologies, i.e., the subdivision of an ontology into small, manageable pieces. This requires two complementary mechanisms: (1) a *clustering mechanism* for grouping a subset of interdependent model components (classes, instances, relations, attributes, and accompanying axioms) into a common module, and (2) an *inclusion* or *import mechanism*, which allows including the model components of

[10] Local range restrictions are typically formulated by means of the universal quantifier (∀).

[11] Postulating the existence of at least one instance of a particular range class is a special case, which can be formulated by means of the existential quantifier (∃).

some ontology module into another module[12]. That way, an ontology can be organized as an inclusion hierarchy of interdependent subontologies.

2.3.6 Notation of Modeling Elements

Having established the major elements of ontology modeling languages, we will now introduce a graphical notation for these elements. This notation, which is based on the UML notation for class diagrams (e.g., Fowler 1997), will be applied throughout this book. Its main components are depicted in Fig. 2.1.

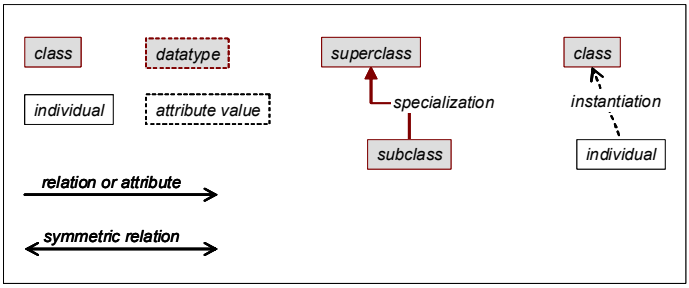

Fig. 2.1: Basic elements for the graphical representation of ontologies

Grey shaded boxes with solid boundary lines represent *classes*, white boxes represent *individuals*. *Datatypes* are denoted by grey shaded boxes with dashed boundary lines, *attribute values* by white boxes with dashed boundary lines. *Specialization* is depicted through a solid line with a solid arrowhead pointing from the subclass to the superclass. A dashed line with an open arrowhead denotes *instantiation*. Binary *relations* are depicted through solid lines, thereby distinguishing three different cases: a line with one open arrowhead represents the standard case of an *unidirectional* relation; a line with two open arrowheads represents a *symmetric* relation; finally, a line without any arrowheads represents a *relation and its inverse* (cf. Fig. 2.2). Please note, that there are further specializations of relations (i.e. for aggregation and composition) which are introduced in detail in Sect. 5.1.3. *Cardinality constraints* are depicted by numbers placed close to the range class of the respective relation. No particular symbols are provided for the other types of axioms.

[12] Inclusion means that if module A includes module B, the model components specified in B are valid in A and can thus be directly used (i.e., extended, refined …) in A. Inclusion is transitive, that is, if module B includes another module C, the ontological definitions specified in C are valid and usable in A, as well.

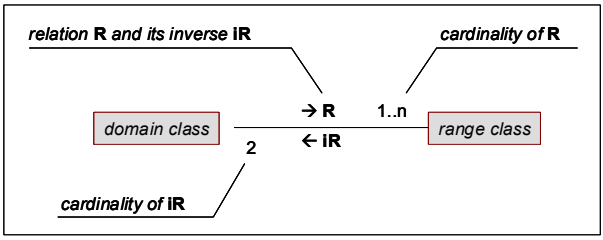

Fig. 2.2: Graphical notation for cardinalities and inverse relations

Generally, classes and relations will be named in accordance with the CamelCase[13] naming convention: UpperCamelCase notation is used to denote identifiers of classes, while relation identifiers are represented in lowerCamelCase notation. No particular naming convention is followed for identifiers of individuals. For better readability, the UpperCamelCase notation is not applied in the text; instead, the individual words that constitute the class identifiers are written separately and in lowercase (e.g., *class identifier*).

2.4 Informal and Formal Specification of an Ontology

Next, we need to discuss the overall form that an ontology must have at the syntactic level in order to be of practical use. An often quoted definition for an ontology stipulates that "an ontology may take a variety of forms, but it will necessarily include a *vocabulary of terms* and some *specification of their meaning*. This includes definitions and an indication of *how concepts are inter-related*" (Uschold et al. 1998). Smith (1996) further postulates that "the ontology should be […] explained in ways which make its content intelligible to human beings, and […] implemented in ways which make this content accessible to computers". From these statements, it can be concluded that two different representations of the ontology are required for practical use, which are referred to as *formal specification* and *informal specification* hereafter[14]. The formal specification is to be processed by AI systems, while the informal specification addresses the human users of the ontology.

– The *formal specification* constitutes an implementation of the ontology in machine-readable form. It specifies the meaning of the vocabulary terms

[13] CamelCase is the practice of writing compound words joined without spaces; each word is capitalized within the compound. While the UpperCamelCase notation also capitalizes the initial letter of the compound, the lowerCamelCase notation leaves the first letter in lowercase.

[14] The informal and the formal specification are different ontologies at the syntactic level, but they represent the same ontology at the semantic level.

and constrains their interrelations (and thus their possible uses) by means of axiomatic definitions, which are stated in a formal modeling language.

– The *informal specification* expresses the definitions of the formal specification in human-readable form. Particularly, it clarifies the meaning of the ontological vocabulary by giving precise term definitions in natural language. Additionally, the interrelations of the terms and their intended usage are described in some appropriate way (e.g., through UML-like diagrams and/or textual descriptions). Some further documentation may be provided, which goes beyond the knowledge stated in the formal specification – for instance, user guidelines for the extension of the ontology.

One of the most common current formal modelling languages is the *OWL Web Ontology Language* (Smith et al. 2004; Bechhofer et al. 2004). OWL and its predecessor *DAML+OIL* (Connolly et al. 2001) are ontology markup languages that have been developed for publishing and sharing ontologies in the Web. Their syntax is based on existing Web markup languages, the most prominent of which is XML (W3C 2006). By now, DAML+OIL has been superseded by its successor OWL, which has been endorsed as a *W3C recommendation*[15]. As OWL is derived from DAML+OIL, it shares most of its features (a listing of the differences between the two languages can be found in Appendix D of Bechhofer et al. 2004). Therefore, only OWL will be discussed in the following.

Model entities are represented through *classes* and *individuals* in OWL. Classes can be hierarchically ordered, thereby allowing multiple inheritances. They can also be further specified through class-embedded axioms stating the *disjointness* of classes, the *equivalence* of classes, or the *extension* of a class. These basic axiom types can be combined by means of the set operators of *union*, *intersection*, and *complement*.

Furthermore, OWL provides language primitives for *attributes* (called 'datataype properties') and *binary relations* (called 'object properties'); higher-arity relations must be represented through classes in OWL. Attributes and relations can be hierarchically ordered, and their usage can be restricted through *range* and *cardinality constraints*. Relations may be further specified through axioms declaring a relation to be *transitive*, *symmetric*, *functional*, or *inverse-functional* (the latter two are also applicable to attributes). Additionally, two distinct relations can be declared to be *equivalent* to, or the *inverse* of, each other. Modularization is supported by the *import* mechanism of OWL, which allows including the definitions and axioms of other ontologies into the current ontology.

The OWL language provides three increasingly expressive sublanguages, called *OWL Lite*, OWL DL, and *OWL Full*. Each of these sublanguages is an extension of its simpler predecessor, both in what can be legally expressed and in what can

[15] A W3C recommendation is the final stage of a ratification process of the World Wide Web Consortium (W3C) concerning a standard for the Web. It is the equivalent of a published standard in other industries.

be validly concluded (Smith et al. 2004). Save for a few exceptions, the representation of OntoCAPE, is restricted to the OWL DL subset. This sublanguage is compatible with a particular type of description logic (DL) called *SHOIN(D)* (Horrocks and Patel-Schneider 2004). As a consequence, the models represented in OWL DL can be processed with standard DL reasoners.
The current release of OWL (version 1.0) lacks certain language constructs, such as those for the relation properties of antisymmetry and reflexivity, or for the representation of qualified cardinality constraints. These (and other) language constructs will be included in the next release of OWL DL, which will move from the *SHOIN(D)* Description Logic to the more expressive *SROIQ(D)* Description Logic (Patel-Schneider and Horrocks 2006).
Rules are currently not part of OWL; however there are plans for an additional rule language that is to be defined on top of OWL (Horrocks et al. 2004). Yet the problem of how to efficiently combine logic-based reasoning and rule-based reasoning still remains to be solved.

2.5 What an Ontology Is and Isn't

With the growing popularity of web-enabled ontology languages like OWL (Smith et al. 2004; Bechhofer et al. 2004), the term 'ontology' is more and more being used in an inflationary manner to denote all kinds of knowledge representation structures. In many of these cases, it is erroneously assumed that the mere use of an ontology modeling language qualifies the respective structure as an ontology. However, this is definitely not the case: Being represented in an ontology modeling language is only a necessary, but not a sufficient criterion for being considered a (formal) ontology.
To better illustrate our point of view, we will below identify two types of ontology-like structures that we do not categorize as full-fledged ontologies: We refer to them as *pseudo ontologies* and *lightweight ontologies*, respectively. In the following, we will define these terms and explain why they do not comply with our – admittedly quite strict – conception of an ontology.
By "pseudo ontology" we mean a part of a software system that is formulated in a formal ontology language such as OWL, but has not been explicitly designed for reuse. A typical example would be the knowledge base of an intelligent system: In our judgment, such a knowledge base – or rather the state-independent part of that knowledge base (cf. Sect. 2.6) –can only be considered an ontology if it is reusable and can thus be shared across software applications and by different groups of

users[16] (cf. Chap. 1). If, on the other hand, the knowledge base has been designed for a single purpose only, we refer to it as a pseudo ontology.
In addition to pseudo ontologies, a second class of ontology-like structures must be differentiated from "true" ontologies. Unlike before, the differentiating factor is not the reusability of the respective structure, but its semantic richness: Structures of this class are not considered full-fledged ontologies as they do not formally define the semantics of the vocabulary terms through axiomatic definitions. Due to their simple internal design, they are sometimes referred to as 'lightweight ontologies' in the literature – as opposed to 'heavyweight ontologies', which model the domain in a deeper way and provide more restrictions on domain semantics (Gómez-Pérez et al. 2004). While a lightweight ontology may be represented in a formal ontology modeling language, it utilizes only a subset of the available modeling elements – that is, a lightweight ontology is built using classes, sometimes instances, and possibly relations, but it does not include relation properties, local constraints, or other forms of axioms. Four types of lightweight ontologies may be distinguished:

- A **controlled vocabulary** is a list of predefined, authorized terms with an unambiguous description given in natural language. The terms may be modeled as classes or instances, but there are no further axiomatic specifications of the meaning of terms.
- A **taxonomy** is a controlled vocabulary that is organized in a hierarchical structure; the hierarchy is usually modeled by means of subsumption relations.
- A **thesaurus** is a taxonomy that additionally specifies certain semantic relationships between its vocabulary terms. Unlike a semantic network (see below), a thesaurus includes only very few types of semantic relationships (typically the synonyms or near-synonyms and the antonyms of a term). These relationships can be modeled through associative relations.
- A **semantic network** is a knowledge representation formalism, which describes terms their relationships in form of a network consisting of labeled nodes and arcs. Typically, the labels of the nodes are nouns, and the labels of the arcs are verbs; that way, the triple formed by two nodes and the interconnecting arc represents a declarative sentence of the form subject-predicate-object. The nodes can be modeled as classes and/or instances, and the arcs can be modeled through associative relations.

Some ontologists (e.g., Guarino 1998; Lassila and McGuinness 2001) prefer a gradual approach to defining ontologies. They do not draw a clear distinction between lightweight and heavyweight ontologies, but postulate an "ontology spectrum" (McGuinness 2002), which ranges from simple taxonomies to sophisticated

[16] This view is supported by numerous ontologists, such as Neches et al. (1991), Borst (1997), Studer et al. (1998), Chandrasekaran et al. (1999), Jarrar and Meersman (2002), Gómez-Pérez et al. (2004), Smith (2006), or Pâslaru-Bontaş (2007).

heavyweight ontologies: Originating from taxonomies, the level of complexity is incrementally increased by adding instances, relations, relation properties, local constraints, and finally global axioms.

2.6 Classification of Ontologies

As the final topic of this theory chapter, a classification framework for ontologies is introduced, and the interdependencies between the different ontology types are discussed. In the later chapters of this book, the classification framework will serve as a frame of orientation to clarify the roles of the individual subontologies that constitute OntoCAPE as well as the roles of those ontologies that are related to OntoCAPE.

According to Guarino (1997b), ontologies can be classified into the following types, which are distinguished by their level of dependence on a particular task or point of view:

- *Top-level ontologies* define general-purpose concepts like object, state, action, etc., which are independent of a particular problem or domain and can therefore be universally applied. In the literature, top-level ontologies are also referred to as *abstract ontologies* (e.g., Borst 1997), *generic ontologies* (e.g., van Heijst et al. 1997a), *foundation(al) ontologies* (e.g., Schneider 2003), or *upper (level) ontologies* (e.g., Guarino 1998). Prominent examples of top-level ontologies are the Top-Elements Classification by Sowa (1995), UpperCyc (Lenat and Guha 1990), or the Suggested Upper Level Merged Ontology SUMO (Niles and Pease 2001).
- *Domain ontologies* capture the knowledge of a domain of expertise, such as medicine or engineering. A domain ontology is not specifically tailored to a particular task or application; instead, it defines general domain knowledge that is relevant for a wide range of different tasks and applications. The goal of a domain ontology is to be universally applicable (and thus reusable) within the respective domain of expertise.
- A *task ontology* (often also referred to as *method ontology*) describes general problem-solving methods that can be applied in different contexts. Such methods are task-specific, but the task itself should be generic in the sense that it occurs in different applications and domains of expertise. An example of a generic task would be graph searching, for which different search methods (e.g., depths-first search or breadth-first search) could be specified in a task ontology. Note that a task ontology does not actually realize (i.e., implement) the method, but only specifies the "terminology for expressing the competence and the knowledge requirements of a method" (Fensel et al. 1996). For a graph searching method, the terminology could, for example, include the concepts of 'current node', 'visited node',

'search depth', etc. Do also note that domain ontology and task ontology have different but complementary objectives with respect to reusability: the former is applicable to different tasks but restricted to a particular application domain; the latter is designated for a particular task but reusable across domains.

– Finally, an *application ontology* provides the concepts that are required for a particular application. To clarify the difference between an application ontology and a knowledge base, Guarino (1997b) proposed the following definition: An application ontology comprises only state-independent information (i.e., facts that are always true), whereas a knowledge base may also hold state-dependent information (i.e., facts and assertions related to a particular state of affairs).

The interdependencies between these four ontology types are depicted in Fig. 2.3: According to Borst (1997) and Guarino (1997b), a task ontology may import the terminology from a top-level ontology and utilize it for the specification of methods. In a similar manner, a domain ontology may describe domain concepts as specializations of the top-level concepts. Furthermore, the concepts in an application ontology can typically be defined by combining and refining concepts from both a domain and a task ontology; this is particularly facilitated if the domain and task ontology are founded on the same top-level concepts and thus share a common world-view. As an example, consider a top-level ontology that introduces the terminology to describe directed graphs. Based on this terminology, a task ontology could specify a graph searching method. Likewise, a domain ontology for chemical engineering could define the concept of a process flowsheets as a special form of a directed graph. An application ontology could finally combine domain knowledge and problem-solving knowledge in order to realize a search application for process flowsheets.

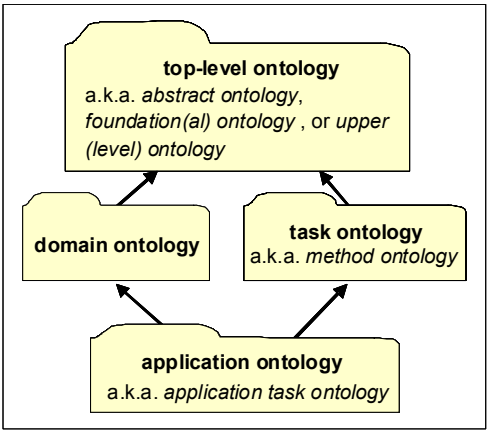

Fig. 2.3: Ontology types and interdependencies according to Guarino (1997b); arrows indicate specialization relationships

While the above classification framework is widely accepted in principle, some points remain subject to debate: The borderline between top-level ontologies on the one hand, and domain and task ontologies on the other hand, is rather vague, as pointed out by van Heijst et al. (1997a); yet, as further argued by these authors, the distinction is intuitively meaningful and useful for building libraries of reusable ontologies. More controversial is the question whether or not it is feasible to separate domain knowledge from knowledge about problem-solving methods (cf. the discussion between van Heijst et al. 1997a, 1997b, and Guarino 1997a). At the core of the discussion is the so-called *interaction problem* (Bylander and Chandrasekaran 1988), which states the following: a method cannot be described without knowing the domain knowledge it will be applied to, and, vice versa, domain knowledge cannot be represented without knowing for what tasks or methods it will be used. Guarino (1997a), while admitting the validity of the interaction problem in principle, argues that one should nevertheless strive for a task-independent representation of domain knowledge; even though the goal cannot be fully achieved, it is quite possible to build a domain ontology that is reusable for a large number of different tasks.

As an extension to the above classification framework, some authors introduce subtypes and combinations of the four basic ontology types:

- Gómez-Pérez et al. (2004) recognize the so-called *general ontologies* (van Heijst et al. 1997a) or *common ontologies* (Mizoguchi et al. 1995) as an additional, distinct type of ontologies. According to these authors, ontologies of this type represent common-sense knowledge that is reusable across domains. However, the differentiating criterion between top-level ontologies and common ontologies remains vague – presumably, a top-level ontology contains only high-level concepts, which must be specialized in domain and task ontologies to become usable, whereas the concepts of a common ontology are directly applicable. A special type of a common ontology would be a *supertheory* – the term has been coined by Borst (1997) to denote an abstract ontology that defines a self-contained theory. Prominent examples of this category are the mereology and topology ontologies created by Borst (1997).
- Some authors (e.g., Mizoguchi et al. 1995; Gómez-Pérez et al. 2004) explicitly subdivide a task ontology in a task part and a method part; only the former part is then referred to as '*task ontology'*, while the latter part is called '*method ontology*'.
- Gómez-Pérez et al. (2004) additionally introduce the type of a *domain-task ontology* which is defined as an application-independent task ontologies that is reusable in a given domain, but not across domains.
- Pâslaru-Bontaş (2007) differentiates between *application domain ontologies* and *application task ontologies*. The former refines and extends the general-purpose knowledge of a domain ontology to the requirements of a particular application, whereas the latter corresponds to a combination of

application-relevant domain and task-related knowledge, similar to the application ontologies introduced by Guarino (1997b).

– Some authors (e.g., Valente and Breuker 1996; van Heist et al. 1997a; Doerr et al. 2003) suggest an additional ontology type called *core ontology*. In the literature, there is no general agreement on what constitutes a core ontology. A core ontology, as we understand it (cf. Brandt et al. 2008a; Morbach et al. 2007; Chap. 12), constitutes the top-level part of an application ontology. More specifically, the function of a core ontology is (1) to select and retrieve the top-level concepts that are relevant for the particular application from the respective domain and task ontologies, (2) to specify how these concepts are to be used (i.e., interpreted) by the application, and (3) to introduce additional top-level concepts required by the application that cannot be retrieved from the available ontologies.

A further type of ontologies, which is not covered by the above classification framework, is the so-called *(knowledge) representation ontology*. Representation ontologies explicate the conceptualizations that underlie knowledge representation formalisms (Davis et al 1993). They are intended to be neutral with respect to world entities (Guarino and Boldrin 1993). That is, they provide a representational framework without making claims about the world (van Heijst et al. 1997a). Top-level ontologies as well as domain and task ontologies are described through the primitives provided by representation ontologies. Well-known examples of this ontology type are the Frame Ontology (Gruber 1993) or the representation ontologies for the Semantic Web languages RDFS (W3C 2000) and OWL (W3C 2002).

Finally, the notion of a *meta model*, or *meta ontology*, needs to be defined. Generally, a meta model is "a design framework, that describes the basic model elements and the relationships between the model elements as well as their semantics. This framework also defines rules for the use […] of model elements and relationships" (Ferstl and Sinz 2001, p. 86). There are two possible interpretations of the term 'meta model' which are consistent with this definition: for their differentiation, Atkinson and Kühne (2002) coined the terms *physical metalevel* and *logical metalevel*. A meta model at the physical metalevel defines the concepts and mechanisms of the modeling language and it thus equivalent to a representation ontology. By contrast, a meta model at the logical metalevel guides the development of the actual ontology by means of predefined types and patterns, which reflect modeling best practice.

Fig. 2.4 presents the extended classification framework, now including both types of meta ontologies[17]. Also, the degree of usability and reusability of the respective ontology types is shown in the figure: Compliant with the usability-reusability trade-off (cf. Sect. 1.3), the usability increases with the ontology type's degree of specialization, whereas its reusability decreases.

[17] The other ontology types introduced above are not depicted since they are merely subtypes of the ones presented in the figure.

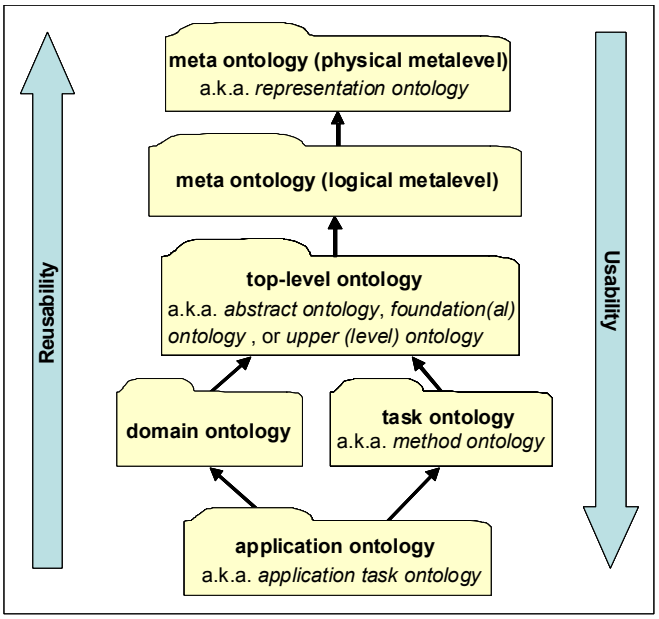

Fig. 2.4: Extended classification framework

2.7 Summary

We have contrasted the similar but different perceptions of 'ontology' in the areas of philosophy and computer science: In the former discipline, an ontology denotes a theory of existence, which may be formulated on any level of formality; it is created for no specific purpose but to gain insight into the respective universe of discourse. In computer science, by contrast, an ontology is created for practical use – either as a shared vocabulary for communication between interacting agents or as a library of reusable knowledge components for building intelligent systems; moreover, a computer science ontology is always formal (and thus machine-interpretable), even though the provision of an additional informal specification for human users is highly advisable.

Over the last decades, several modeling paradigms and modeling languages have been proposed for the representation of formal ontologies. We have presented the common elements and pointed out the differences of these paradigms and languages.

Different types of ontologies can be differentiated: Firstly, one needs to distinguish between full-fledged 'heavyweight' ontologies and 'lightweight' ontologies, which do not make use of axiomatic definitions. Secondly, one must distinguish truly reusable ontologies from 'pseudo ontologies', which are built for a single application only. Finally, an ontology may be partitioned into sub-ontologies of different types, which can be classified according to their respective functions; the most common types, ordered by increasing usability, are meta ontology, top-level ontology, domain ontology, task ontology, and application ontology.

2.8 References

Atkinson C, Kühne T (2002) The role of metamodeling in MDA. In: *Proceedings of the Workshop in Software Model Engineering (in conjunction with UML'02, Dresden, Germany).* Online available at http://www.metamodel.com/wisme-2002/papers/atkinson.pdf. Accessed January 2008.

Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (2003) *The Description Logic Handbook: Theory, Implementation, Applications*. Cambridge University Press, Cambridge.

Bechhofer S, van Harmelen F, Hendler J, Horrocks I, McGuinness D, Patel-Schneider L, Stein LA (2004) *OWL Web Ontology Language Reference*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-ref/. Accessed September 2007.

Borst WN (1997) Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD Thesis, Centre for Telematics and Information Technology, University of Twente.

Brandt SC, Fritzen O, Jarke M, List T (2008a) Goal-oriented information flow management in development processes. In: Nagl M, Marquardt W (eds.): *Collaborative and Distributed Chemical Engineering*. Springer, Berlin:369–400.

Bylander T, Chandrasekaran B (1988) Generic tasks in knowledge-based reasoning: the right level of abstraction for knowledge acquisition. In: Gaines B, Boose J (eds.): *Knowledge Acquisition for Knowledge-Based Systems*. Academic Press, London:65–77.

Chandrasekaran B, Josephson JR, Benjamins VR (1999) What are ontologies, and why do we need them? *IEEE Intell. Syst.* **14** (1):20–26.

Chen PP (1976) The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems* **1** (1):9–36.

Connolly D, van Harmelen F, Horrocks I, McGuinness DL, Patel-Schneider PF, Stein LA (2001) *DAML+OIL reference description*. W3C Note, 18 December 2001. Online available at http://www.w3.org/TR/daml+oilreference. Accessed January 2008.

Davis R, Shrobe H, Szolovits P (1993) What is a knowledge representation? *AI Mag.* **14** (1):17–33.

Doerr M, Hunter J, Lagoze C (2003) Towards a core ontology for information integration. *J. Digit. Inf.* **4** (1), Article No. 169.

Fensel D, Schönegge A, Groenboom R, Wielinga BJ (1996) Specification and verification of knowledge-based systems. In: Gaines BR, Musen MA (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SRDG Publications.

Ferstl OK, Sinz EJ (2001) *Grundlagen der Wirtschaftsinformatik, Bd. 1*. Oldenbourg, München.

Fowler M (1997) UML Distilled – Applying the Standard Object Modeling Language. Addison-Wesley.

Gómez-Pérez A, Fernández-López M, Corcho O (2004) *Ontological Engineering*. Springer, Berlin.

Gruber TR (1993) A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.* **5** (2):199–220.

Gruber TR (1995) Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum Comput Stud.* **43** (5/6):907–928.

Gruber TR, Olsen GR (1994) An Ontology for Engineering Mathematics. In: Doyle J, Torasso P, Sandewall E (eds.): *Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*. Morgan Kaufmann. Online available at http://www-ksl.stanford.edu/knowledge-sharing/papers/engmath.html. Accessed September 2007.

Guarino N (1997a) Understanding, building, and using ontologies: A commentary to "Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber, and Wielinga. *Int. J. Hum Comput Stud.* **46** (2/3):293–310.

Guarino N (1997b) Semantic matching: formal ontological distinctions for information organization, extraction, and integration. In: Pazienza MT (ed.): *Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*. Springer, Berlin:139–170.

Guarino N (1998) Formal ontology and information systems. In: Guarino N (ed.): *Formal Ontology in Information Systems*. IOS Press, Amsterdam:3–15.

Guarino N, Boldrin L (1993) Ontological requirements for knowledge sharing. In: Skuce, D. (ed.): Proceedings of the IJCAI'95 Workshop on Basic Ontological Issues in Knowledge Sharing.

Guarino N, Giaretta P (1995) Ontologies and knowledge bases: towards a terminological clarification. In: Mars N (ed.): Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing. IOS Press, Amsterdam:25–32.

Hodges W (1983) Elementary predicate logic. In: Gabbay DM, Guenthner F (eds.): Handbook of Philosophical Logic – Vol. I: Elements of Classical Logic. Reidel, Dordrecht:1–131.

Hofweber T (2005) Logic and ontology. In: Zalta EN (ed.): The Stanford Encyclopedia of Philosophy (Winter 2005 Edition). Online available at http://plato.stanford.edu/achives/win2005/entries/logic-ontology/. Accessed January 2007.

Horrocks I, Patel-Schneider P (2004) Reducing OWL entailment to description logic satisfiability. *J. Web Sem.* **1** (5):345–357.

Horrocks I, Patel-Schneider P, Boley H, Tabet S, Grosof B, Dean M (2004) *SWRL: A Semantic Web Rule Language Combining OWL and RuleML.* W3C Member Submission 21 May 2004. Online available at http://www.w3.org/Submission/SWRL/. Accessed December 2007.

Jarrar M, Meersman R (2002) Scalability and knowledge reusability in ontology modeling. In: *Proceedings of the International conference on Infrastructure for e-Business, e-Education, e-Science, and e-Medicine* SSGRR2002.

Lassila O, McGuinness D (2001) The Role of Frame-Based Representation on the Semantic Web. Technical Report (KSL-01-02), Knowledge Systems Laboratory, Stanford University. Online available at http://www-ksl.stanford.edu/KSL_Abstracts/KSL-01-02.html. Accessed October 2007.

Lenat D, Guha RV (1990) Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project. Addison Wesley.

Lloyd JW (1987) Foundations of Logic Programming, 2nd edition. Springer, Berlin.

McGuinness DL (2002) Ontologies come of age. In: Fensel D, Hendler J, Lieberman H, Wahlster W (eds.): *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*. MIT Press:171–194.

Minsky M (1975) A framework for representing knowledge. In: Winston PH (ed.): *The Psychology of Computer Vision*. McGraw-Hill, New York.

Mizoguchi R, Vanwelkenhuysen J, Ikeda M (1995) Task ontologies for reuse of problem solving knowledge. In: Mars N (ed.): *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing*. IOS Press, Amsterdam:46–57.

Morbach J, Yang A, Marquardt W (2007): OntoCAPE – a large-scale ontology for chemical process engineering. *Eng. Appl. Artif. Intell.* **20** (2):147–161.

Neches R, Fikes R, Finin T, Gruber T, Patil R, Senator T, Swartout WR (1991) Enabling technology for knowledge sharing. *AI Mag.* **12** (3):36–56.

Newell A (1982) The knowledge level. *Artif. Intel.* **18** (1):87–127.

Niles P (2001) Towards a standard upper ontology. In: Guarino N, Welty C, Smith B (eds.): Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS-2001). ACM:2–9.

Pâslaru-Bontaş E (2007) *Contextual Approach to Ontology Reuse: Methodology, Methods and Tools for the Semantic Web.* PhD Thesis, FU Berlin.

Patel-Schneider PF, Horrocks I (2006) *OWL 1.1 Web Ontology Language Overview*. W3C Member Submission, 19 December 2006. Online available at http://www.w3.org/Submission/owl11-overview/. Accessed October 2007.

Schneider L (2003) How to build a foundational ontology: the object-centered high-level reference ontology OCHRE. In: Günter A, Kruse R, Neumann B (eds.): *KI 2003: Advances in Artificial Intelligence*. Springer, Berlin:120–134.

Smith B (1996) Mereotopology: a theory of parts and boundaries. *Data Know. Eng.* **20** (3):287–303.

Smith B (2006) Against idiosyncrasy in ontology development. In: Bennett B, Fellbaum C (eds.): *Formal Ontology in Information Systems*. IOS Press:15–26.

Smith MK, Welty C, McGuinness DL, eds. (2004) *OWL Web Ontology Languages Guide*. W3C Recommendation, 10 February 2004. Online available at http://www.w3.org/TR/owl-guide/. Accessed October 2007.

Sowa JF (1995) Top-level ontological categories. *Int. J. Hum Comput Stud.* **43** (5/6):669–685.

Studer S, Benjamins VR, Fensel D (1998) Knowledge engineering principles and methods. *Data Knowl. Eng.* **25** (1/2):161–197.

Uschold M, Grüninger M (1996) Ontologies: principles, methods and applications. Knowl. Eng. Rev. 11 (2):93–136.

Uschold M, King M, Moralee S, Zorgios Y (1998) The enterprise ontology. *Knowl. Eng. Rev.* **13**:31–89.

Valente A, Breuker J (1996) Towards principled core ontologies. In: Gaines BR, Mussen M (eds.): *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. SDRG Publications.

van Heijst G, Schreiber AT, Wielinga BJ (1997a) Using explicit ontologies in KBS development. *Int. J. Hum Comput Stud.* **46** (2/3):183–292.

van Heijst G, Schreiber AT, Wielinga BJ (1997b) Roles are not classes: a reply to Nicola Guarino. *Int. J. Hum Comput Stud.* **46** (2/3):311–318.

W3C (2000) *The RDFS representation ontology*. Web resource. Online available at http://www.w3.org/2000/01/rdf-schema. Accessed June 2008.

W3C (2002) *The OWL representation ontology*. Online available at http://www.w3.org/2002/ 07/owl. Accessed October 2007.

W3C (2006) *Extensible Markup Language (XML)*. Online available at http://www.w3.org/XML/. Accessed December 2007.