


How Ontologies Can Help in Software Engineering

Cesar Gonzalez-Perez 

Incipit CSIC, Avda. de Vigo, s/n, 15705 Santiago de Compostela, Spain
cesar.gonzalez-perez@incipit.csic.es

Abstract. Ontologies are often understood as having a historical background quite different to that of software engineering, which has caused a number of issues when trying to use them in this context. However, recent works have characterized ontologies as being closely related to models and metamodels, thus allowing for an inclusive treatment and use. In this work I describe how ontologies are understood today within software engineering, how they relate to models and metamodels, and how they are useful to software and systems engineering over different lifecycle phases, in different domains, and in relation to standards such as those from ISO/IEC JTC1 SC7.

Keywords: Ontologies · Models · Conceptual modelling · Domain-specific modelling · Metamodelling · Modelling languages · Standards

1 Context and Motivation

In philosophy, *ontology* is the science of being [82]. That is, ontology is the branch of philosophy concerned with the study of what is and how it is, regardless of what we may know about it. As a countable noun, *an ontology* is a theory of the world in terms of what exists and how, again regardless of what our knowledge about it may be. In turn, the study of knowledge (including belief, truth and justification) is part of *epistemology*, a different branch of philosophy [83 “Epistemology”].

Outside philosophy and in the computing realm, the word “ontology” is often used with a different but related meaning; in 1993, Gruber [28], a researcher in artificial intelligence and knowledge engineering, famously defined an ontology as “a formal specification of a conceptualization”. There are two especially relevant differences between this definition and the philosophical view described above:

- In computing, ontologies are formal, i.e., they are expressed in a formal or at least semi-formal language, through which relevant concepts and their properties and relationships are captured. In philosophy, however, ontologies are often presented in natural language; see e.g. [10].
- In computing, an ontology captures a conceptualization, that is, some knowledge as held by one or more people. This brings computing ontologies closer to epistemology (the science of knowledge) than philosophical ontology (the science of knowledge-independent being), as noted by [31].

Despite these differences, we can still claim that computing ontologies are theories of (a part of) the world, because they describe what a portion of reality looks like (although many anti-realists would disagree). In this regard, and according to Gregor [27], they are analytical (i.e., type 1) theories, since they describe the target domain without attempting to explain causal phenomena or predict future observations.

Two additional points must be made in relation to Gruber's definition and its context. First of all, the definition provided by Gruber, and especially the way in which it has been applied since, correspond mostly to what today we call *domain ontologies*, i.e., ontologies that describe a particular subject or field of work such as genomics [2], lexical structures [66] or cultural heritage [52]. In addition to domain ontologies, however, the literature has also studied *upper ontologies*, which, rather than describing a specific area of reality, aim to establish what reality is like in general, making statements that should be valid for any domain or application. For example, a domain ontology about lexical structures may contain concepts such as "Determiner" or "Adjective", which are only relevant within that domain; similarly, a domain ontology for genomics may contain concepts such as "Gene" or "Transposon". An upper ontology, however, may contain concepts that describe the very fabric of reality, such as "Type", "Property" or "Role"; see, e.g., BORO [76], DOLCE [62] or UFO [34]. The relationship between domain ontologies and upper ontologies is that of *conformance*, i.e., a domain ontology conforms to a particular upper ontology, very much as a model conforms to a metamodel. This fact and its consequences are explored in Sect. 2.

Secondly, Gruber, as well as other authors who adopted and further developed the application of ontologies in computing [29, 32, 38], were part of the artificial intelligence (AI) community, which since the 1970s had been pressed to find a good manner to describe states of affairs in the world on which machines were able to apply automatic reasoning. Agent technologies, which became a fashionable research theme in the late 1990s and early 2000s, also pushed in this direction [12], and ontologies (as in computing rather than philosophy) were developed as a solution to this problem. At the same time, the software engineering community was developing solutions to cope with the increasing complexity of the systems that were being developed, and introducing a wide range of modelling languages and approaches over the 1980s and 1990s [14, 26, 61, 79]. As a result, the same problem (namely, representing the world in a suitable fashion) was being addressed by two communities at the same time and with not much exchange of information [46], and focusing on very different aspects of the problem. As pointed out by [31], "AI researchers seem to have been much more interested in the nature of reasoning rather than in the nature of the real world". The solutions thus obtained by the two communities (ontologies and modelling approaches) share some commonalities, but also differ in significant ways, which are explored in Sect. 2. In particular, in software engineering we use models for two major purposes: to describe domains and to specify systems [21]. A model used to describe a domain looks much like a domain ontology, and the relationship between models and ontologies is also explored in Sect. 2.

Finally, it is worth mentioning that ontologies have become quite typical in the semantic web [7] field, particularly through World Wide Web Consortium (W3C) specifications such as the Web Ontology Language (OWL) [86] and the Simple Knowledge Organization System (SKOS) [85]. SKOS and most especially OWL use a very

computation-oriented notion of what ontologies are and focus on web-based solutions, which introduces significant “implementation noise” that makes these approaches barely usable outside their niche. This is also discussed in Sect. 2.

Over the following sections, I will focus on how ontologies can be useful in software engineering; for this reason, the discussion will be centred on software engineering and, in general, I will use software engineering terms and concepts. Terminology and concepts that are specific of the ontologies field are explicitly flagged.

In particular, the following sections describe three major aspects on how ontologies can be useful in software engineering. First, ontologies help obtain a better philosophical grounding of the software engineering discipline and practice, solving some issues that often pass unnoticed. Second, ontological thinking allows us to carry out better domain modelling by contributing aspects often neglected by traditional modelling technologies. Third, ontologies constitute an excellent basis for the standardisation of the software engineering field, especially within the work carried out by organizations such as ISO.

2 Ontologies and Models

Ontologies are intuitively close to models in software engineering, as described above. However, before I discuss ontologies and their relationship to models, some concepts must be fixed. This section provides some base concepts and terms, and then discusses the differences and commonalities between ontologies and models.

2.1 Base Concepts in Models, Metamodels and Modelling Languages

A **model** is “an abstraction that represents some view on reality, necessarily omitting details, and for a specific purpose” [39], or “an abstraction of a (real or language-based) subject allowing predictions or inferences to be made” [59, 60], or “a statement about a given subject under study (SUS), expressed in a given language” [23], or even “a description of (part of) a system written in a well-defined language” [18]. In any case, a model always involves the following [43]:

- Something that is represented, i.e., the modelled subject. (Mapping)
- An abstraction process, which eliminates irrelevant details of the former to keep only what is relevant to a particular purpose. (Simplification)
- An ability to reason on the model and then apply the conclusions of the reasoning to the modelled subject, i.e., a proxy function. (Application)

As seen above, representation plays a central role in models. A model can represent the subject through mappings of three different kinds [21]:

- **Isotypical**, by which an element in the model maps straightforwardly to an entity in the modelled subject. For example, an architectural plan of a house usually represents the real house isotypically, since it maps to that house and only that house. Also, an object in an object-oriented model or running process usually represents the real entity it refers to isotypically.

- **Prototypical**, by which an element in the model maps to a set of entities in the modelled subject given by example; in other words, the element in the model exemplifies the kind of subject entities that are being represented. For example, a model car placed next to a cardboard model house to illustrate where cars are expected to park represents cars prototypically, since the model car does not map to any particular real car, but just to an example car.
- **Metatypical**, by which an element in the model maps to a set of entities in the modelled subject given declaratively; in other words, the element in the model is a description of the properties that subject entities must comply with in order to be represented. For example, the technical specifications of the windows to install in the house from our previous example constitute a metatypical representation, since they do not depict a specific window or exemplify a set of allowed windows, but declare what properties any window must possess in order to be acceptable. A class in an object-oriented model or computer program also represents the real entities it refers to metatypically.

Models that work in an isotypical manner have been called in the literature token models, and those who represent metatypically have been called type models [60]. This distinction is old, having been introduced by philosopher Charles Sanders Peirce in the late 19th century, and plays an important role in contemporary ontological thinking [83 “Types and Tokens”]. However, and since different elements in one model can work in different manners (isotopically, prototypically or metatypically), I prefer the more precise granularity of the latter rather than the simplistic classification into token and type models.

A **metamodel**, in turn, is a particular kind of model, as indicated by the qualifier “meta-”; a metamodel is a “model of models” [68] or “a model of a set of models” [18]. Either case, it is clear that a metamodel is a model for which the modelled subjects are also models. The relationship between a metamodel and the models that it represents is one of conformance [39], i.e., a model *conforms to* a metamodel.

Also, and very importantly, since metamodels are a specific type of models, everything that we state about models also applies to metamodels, including their ability to represent their subjects (i.e., other models) isotypically, prototypically or metatypically.

Defining what a **modelling language** is proves harder. For some authors, a modelling language is “a set of models” [18], i.e., a language is the set of all possible models that may be possibly expressed in that language. According to this view, a specification (or model) of that language constitutes a metamodel, since we said that a metamodel is a model of a set of models. This is analogous to saying that English is the set of all possible sentences that may be possibly uttered in this language, and that a specification (or model) of English constitutes its grammar (cf. metamodel).

Other authors, however, place no emphasis in this difference between metamodels and languages, and define a modelling language as “an organised collection of model unit kinds that focus on a particular modelling perspective” [23, 56 clause 7.1.18], where model unit kinds are the primitives that this language uses to express models, e.g., “Class” or “Association” in UML [69].

2.2 Base Concepts in Ontologies

As stated earlier, an **ontology** is “a formal specification of a conceptualization” [28, 29], or “a formal, explicit specification of a shared conceptualization” [15]. Also, “an ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members)” [30]. However, it is often emphasised in the literature that ontologies do not need to be composed of type-level elements only, and they may also contain instance-level elements such as objects, often called “individuals” in ontology parlance [15, 86], as well as axioms [15, 86] that further constrain the semantics of the involved types and instances.

Furthermore, ontologies are usually described as containing knowledge rather than data [28], that is, they work at the *knowledge level*, a concept introduced by [67] in the 1980s. Knowledge and data, together with the intermediate level of information and the top level of wisdom, compose the Ackoff “pyramid” [1] of increasing abstraction. Thus, by representing the world in terms of knowledge rather than data, ontologies are supposed to be more abstract than, say, database schemata, and provide better support for semantics, especially in the context of the semantic web [7]. According to [39], ontologies were introduced and popularised within the software engineering community from the early 2000s and onwards, as shown by the increasing literature on the subject, the availability of specific tools (such as Protégé protege.stanford.edu or Swoogle-woogle woogle.umbc.edu) and ontology repositories, and the number of projects devoted to ontologies. Still, some authors have pointed out that the promise of semantic knowledge, especially on the web, is still unrealised [84].

Another essential aspects of ontologies in computing, hinted at above, is that they must be formal and, more precisely, understandable by a computer or “codified in a machine interpretable language” [15]. In fact, automatic (i.e., algorithmic) reasoning is often presented as a key motivation to develop ontologies [31, 86]. To this purpose, ontology languages such as CycL [33] or OWL [86] have been developed that focus on rigorous implementation of formal logic. The amount of detail required to create an ontology, as well as the associated “implementation noise”, are usually quite large; this is a contradiction with the principle of minimal encoding bias [29], which states that a good ontology should be expressed at the knowledge level and be as free from encoding details as possible. In addition, this means that creating an ontology by hand (on paper or on a whiteboard, for example) and dynamically exploring alternatives is extremely difficult, and specialised tools are obligatory.

Finally, a clear distinction must be made between upper, or foundational, ontologies and domain ontologies, as introduced in previous sections. An **upper ontology** is an “axiomatic account of high-level domain-independent categories about the real world” [80], or one that “defines a range of top-level domain-independent ontological categories, which form a general foundation for more elaborated domain-specific ontologies” [36]; this means upper ontologies should be valid across domains and contain very abstract concepts only. In turn, a **domain ontology** is a “specific theory about a material domain (e.g., law, medicine, archaeology, molecular biology, etc.)” or “a shared

conceptual specification of the domain” [34]. Developing a domain ontology requires a deep understanding of the particular domain of application; however, developing an upper ontology requires a deep understanding of reality and the commitment to specific meta-ontological choices as exemplified by [75], such as the nature of categorisation or the structure of time.

This has several consequences. Firstly, it seems that upper ontologies closely match the field of study of philosophical ontology, whereas domain ontologies are closer to epistemology, since they describe a domain in terms of human-mediated knowledge [31]. Secondly, upper ontologies establish a structure to which domain ontologies can conform, by serving as a starting point to build new (domain) ontologies, as a reference for the comparison of different (domain) ontologies, and as a common framework for (domain) ontology harmonisation and integration [62].

2.3 Differences Between Ontologies and Models

As discussed above, ontologies and models seem to be trying to address the same problems (representing the world in an abstract manner) but do it from very different perspectives. These differences often result in different artefacts, different uses and different possibilities:

- Ontologies are intended for computer processing, whereas models are aimed at human understanding (but see below).
- Ontologies are highly formal and require a logical basis, whereas models can be semi-formal and admit some degree of informality.
- Ontologies are harder to develop, whereas some models can be created quite easily.
- Ontologies aim to represent the world objectively, as it is, whereas models are inherently subjective.
- Ontologies often combine type (i.e., metatypical) and token (i.e., isotypical) representations together, whereas models tend to emphasise the difference.

First of all, the overall motivation for ontologies has been automated, algorithmic reasoning [31, 86] carried out by machines. This has meant that an ontology is usually a computer-oriented artefact, not always easily readable by humans. Contrarily, modelling in software engineering has been motivated since the 1980s by the need to tackle complexity and understand better the world around us as well as obtain better specifications for the systems that engineers will build [44]. This means that models are usually human-oriented artefacts that machines cannot process directly. However, the model-based software engineering (MBSE) approach [9, 81], popularized in the last 15 years, has changed this significantly. These days, models are often constructed as machine-readable artefacts that can be processed by a computer to generate other models or even code through MDA/MDE approaches [68] or languages such as Executable UML [64]. Still, much modelling is still not machine-based and oriented towards humans. From the ontologies side, work in ontology visualization [58] is being carried out to make ontologies more easily understandable to humans. In summary, ontologies and models have very different historical aims, which are now converging.

Since ontologies are traditionally geared towards computers, they are often based on some form of formal logic, and an ontology, as an artefact, is a highly formal one. This is particularly noticeable when looking at ontology languages; for example, CycL [33] is based on first-order logic and has some support for modal operators and higher-order quantification (such as “all” and “exists”); similarly, OWL [86] is a “computational logic-based language” that supports full algorithmic decidability in its OWL-DL (description logic) variant. Contrarily, many modelling languages rarely aim to attain full formality, with the exception of those particular to the formal methods subfield or oriented towards MBSE. Modelling languages, in general and as usually employed in software engineering, are based on meta-specifications such as MOF [70] that make extensive use of natural language and thus leave room for informality. Again, this is changing now, and implementations based on languages such as UML are being successfully used for machine processing.

As a further consequence, ontologies are usually harder to develop than models. An example of this is the fact that ontologies usually require great care when identifying and naming classes; in OWL, for instance, a class is identified by an international resource identifier (IRI), which must be correctly generated and namespaced. In UML [69], however, a class is identified by a simple name in natural language. For reasons like this, it is very easy to informally sketch an exploratory model on a piece of paper or a whiteboard, but it is very hard to do this for an ontology. However, the ontology, once created, will have a degree of formality and a potentiality for automatic processing that the model may lack.

As an additional major difference, ontologies aim to represent the world objectively, as it is, without introducing much subjective bias, whereas models may embrace subjectivity. This is particularly so in the case of upper ontologies, although domain ontologies, given their focus on shared conceptualizations [15], also have this property. According to [31], ontologies constrain the meanings they aim to provide (through axioms, for example), whereas conceptual models offer a fully subjective and pre-interpreted view of the represented subject. In the case of upper ontologies, this is even more so, as illustrated in [80] when describing foundational (i.e., upper) ontologies as being related to “reusable information”, “semantic interoperability” and “axiomatic accounts of high-level domain-independent categories”. In modelling, to the contrary, a very specific purpose is always taken as a starting point, and it is assumed that this purpose strongly shapes the resultant model; as pointed out by [4], “software engineers have taken a very pragmatic approach to data representation, encoding only the information needed to solve the problem in hand”. Also, the statistician George Box is usually credited as the author of the famous aphorism “All models are wrong; some models are useful”; this is often interpreted to mean that models, given the fact that they represent through abstraction, are necessarily discarding details, and are therefore “wrong” or biased in some way as dictated by the guiding purpose [43].

Lastly, ontologies often emphasise that a good account of reality is given by combining classes and instances in the same representation, and usually there is no particular emphasis in differentiating layers or levels. The modelling community, however, has developed strong ideas about the separation of type (i.e., metatypical) and token (i.e., isotypical) representations, such as OMG’s strict metamodelling paradigm [3] and,

although classes and objects can be mixed together in the same models in, for example, UML, this is very rarely done.

Additional differences between ontologies and models are reported and discussed in [4].

2.4 Commonalities of Ontologies and Models

Despite the differences described in the previous section, numerous works have tried to find commonalities between ontologies and models. This is not surprising, since, as pointed out above, models and ontologies are trying to solve much the same problems, and some common grounds are to be expected. In addition, cross-pollination between disciplines is often seen as a motivation.

In [4], the authors characterize models and ontologies over several key aspects, and observe that “all ontologies are models, but not all models are ontologies”, since any information representation that fulfils the necessary conditions to be an ontology also fulfils those to be a model. This means that ontologies are a specific kind of models and that, therefore, everything we say about models should also apply to ontologies. Also, the authors convincingly criticise many of the claims that are usually employed to highlight the differences between models and ontologies. For example, they show that support for reasoning is not a definitional property of ontologies, that there is no requirement for open or closed world assumptions for either models or ontologies, and that it is perfectly possible to create information representations that are not shared (and therefore are not ontologies) using ontology languages. All these facts mean that ontologies and models are extremely similar, much more than often depicted. However, and although the authors state that these strong similarities make many ontology-driven efforts and technologies redundant, this is hard to sustain, since a subtype usually adds details to the super type it derives from, and hence ontologies are likely to possess specific properties (such as those described in Sect. 2.3) that are not present in models. Still, most of the observations in [4] are valid and constitute a strong change of direction to the usual discourse and its emphasis on difference.

A few years later, [39] tackled similar concerns from a different angle. Here, the author relates models to domain ontologies, and metamodels to upper (foundational) ontologies. In both cases, the author points out that other works also coincide in equating or relating domain ontologies and models, such as [19, 47], and upper ontologies with metamodels [34, 35].

It thus seems that ontologies and models, despite being often presented as different technologies, are not that different after all. This is compatible with our experience when, in 2006, we “extracted” a domain ontology for software development methodologies [22] from an existing model of the same domain [56] with little effort. Apparently, the same representation could be easily cast as either a software engineering model or a domain ontology; this made us realise that ontological thinking may be applicable to software engineering as a fruitful contribution.

3 Using Ontologies

Previous sections have described ontologies, models, and the relationships between them, focussing on differences and similarities. At the end of the last section, I concluded that ontologies and models are not too different, and that, for this reason, bringing over ontological thinking into software engineering should be feasible. In this section, I explore three major areas where ontologies have proven useful to software engineering over the last few years: philosophical grounding, domain modelling and standardisation.

3.1 For Philosophical Grounding

It is interesting to observe how software engineering has focussed so much in representing reality, but invested so little in understanding the implications of these representations [46, 77]. Often, we make representational choices without being too conscious of the consequences, and some choices are never made because we cannot even think of them. Philosophy, however, has been dealing with the issue of representing reality for some time, and can help. Thus, the philosophical grounding of modelling has become the theme of some recent works in software engineering, in which ontologies (especially upper) play an important role.

My colleagues and I have devoted some time to searching for answers to questions such as “What are conceptual models made of?”, “What do classes in class models actually represent?” or “What is the relationship between conceptual models, mental models and physical reality?” [46, 77]. Take, for example, the second question. Assuming that classes in class models represent categories of things, often called “universals” in philosophy, do they stand for universals-as-they-are or rather universals-as-we-know-them? In other words, do classes directly represent things in the world (ontological, direct representation) or do they represent mental concepts, which in turn represent things in the world (epistemic, mediated representation)? If the latter, and assuming that mental concepts may be different from an individual to the next, how are we sure that a class in a model stands for the “right” concept? How do we eliminate subjectivity and ambiguity so that a shared understanding is achieved?

This line of reasoning has also been used to analyse specific aspects of modelling, such as whole/part relationships in object-oriented models [41, 73, 74] or the UML itself [72]. In [41], the authors characterize whole/part relationships by ontological analysis and describe a number of primary (necessary, Boolean) and secondary (classificatory, not necessarily Boolean) characteristics of these relationships. In [73, 74], the authors continue to differentiate resultant and emergent properties by using Bunge’s ontology [10, 11]; a resultant property is a property of an aggregate that is a direct result of properties of its parts (the whole equals the sum of its parts), whereas an emergent property of an aggregate is one that is not provided by any properties of its parts, but rather emerges from their interaction (the whole is greater than the sum of its parts). For example, a car engine is an aggregate of individual mechanical parts: the engine has a resultant “Weight” property, directly obtained from its members’ properties, as well as an emergent “Peak Power” property, which materialises from the interactions of its members rather than being contributed directly by the members’ properties. The authors

in [73, 74] conclude that an aggregate (the “whole” in a whole/part relationship) must possess at least a resultant and an emergent property; otherwise, it would not be a true aggregate.

A similar ontological analysis based on the Bunge-Wand-Weber approach [78] has been carried out by [72] on the UML itself, resulting in a comprehensive set of recommendations to enhance UML. Some of the improvement areas include:

- Distinguishing between physically impossible and humanly disallowed events.
- Achieving better separation between the description of the domain and the specification of the system.
- Introducing additional modelling primitives to avoid overloading, i.e., the fact that some existing modelling constructs are used for several different purposes.

Precisely, ontological analysis has been especially useful to explicitly clarify and solve some obscure areas of modelling. For example, it has been long known that the “is-a” construct in modelling was being used with little rigour to represent very different semantics; in fact, [32] discusses the problem of “ISA overloading” back in 1998, and proposes an initial framework to avoid it. We have observed that the problem is compounded by the fact that the copula *to be* in English, very much like in most other Indo-European languages, is extremely overloaded with meaning. We have identified at least five senses in which the verb *to be* is regularly used in the modelling literature:

- **Existence**, by which something is said to exist, e.g., “There is a person”.
- **Identity**, by which two entities are said to be the same, e.g., “Isabel is my wife”.
- **Predication**, by which a property is associated to an entity, e.g., “Isabel is tall”.
- **Classification**, by which an entity is assigned to a type or class, e.g., “Isabel is a person”.
- **Generalisation**, by which a type or class is said to be subsumed by a more abstract one, e.g., “A person is a living being”.

In modern-day object oriented languages, existence of an entity is conveyed by the existence of the corresponding object; identity is not conveyed but delegated to the real-world entity; predication is easily conveyed through attribute values; classification is conveyed through the object’s “instance-of” relationship towards its class; and generalisation is conveyed through generalisation/specialisation relationships between classes. Thus, I do not see any problem with “is-a” overloading today as long as a well-defined language is used that supports object identification, attribute values, instantiation relationships and generalisation/specialisation relationships as separate modelling primitives.

Ontological reasoning is sometimes confronted with linguistic or epistemic thinking, especially when discussing alternative ways of representing. In [5], for example, “logical” and “physical” representations are described: when we say that a particular book object in a library management system *is a book*, we are using a logical representation; when we say that this object *is an object*, we are using a physical representation. As discussed by [23], physical models represent ontologically, using concepts from what we have called upper or foundational ontologies, such as “Object”. Contrarily, logical models represent epistemically, using concepts from what we have called domain ontologies, such as “Book”. Although some authors insist that physical and logical modelling

(sometimes confusingly named linguistic and ontological modelling, respectively, such as in [6]) are orthogonal manners of representing the same reality, it is easily seen that they are not, and in fact logical models conform to physical models, very much like domain ontologies conform to upper ontologies, and therefore a linear (rather than orthogonal) chain of models arises as proposed by [23].

An additional area where ontological thinking has been used in software engineering is that of language development. ConML is a conceptual modelling language designed for users with no previous exposure to information technologies and especially oriented towards domains in the humanities and social sciences [20, 48]. Although ConML superficially resembles UML, it contains some aspects that are worth mentioning. One of them is that of symmetric unary associations. Most associations are binary (i.e., they link two types together) or even higher-arity, but some are unary, which link a type back to itself. Of these, some entail an asymmetric relation between the instances they connect, whereas others establish a symmetric relation. UML and other conventional languages provide no support to model this latter kind of associations, despite being extremely common in real life: for example, a place and its neighbouring places, a person and his/her spouse, an author and his/her co-authors, a mathematical function and its inverse, an archaeological-site and all those others that are visible from it. Since these associations involve a single role (for both “ends”) attached to a single class, and UML requires that every association end attached to a type has a different qualified name, these associations cannot be expressed in UML. The solution adopted by ConML is straightforward, namely allowing for associations with a single “end” [48 clause 5.6.9], and its novelty does not reside so much in the adopted solution as in the detection of the need and the insight to differentiate between symmetric and asymmetric unary associations.

Also in relation to ConML, ontological thinking allowed us to improve the usual treatment of null semantics that is found in most languages. Usually, “null” means no data, but no distinction is made between ontological and epistemic reasons for this absence. For example, if the “Name” column in a “Persons” table contains “null” for a particular row, does this mean that this person lacks a name (ontological absence) or rather that we do not know it (epistemic absence)? This is easy to determine for some properties, which by nature cannot be ontologically absent (e.g., “Age” in the above mentioned table), but impossible for others. For this reason, ConML uses *null* to indicate ontological absence of information (i.e., “this data does not exist”) and *unknown* to indicate epistemic absence of information (i.e., “this data exists but we do not know about it”) [44 Problem 5, 48 clause 5.6.8]. This allows for more precise semantics and a better representation of the domain.

3.2 As Domain Models

Regarding the second area of ontology use in software engineering, it is worth noting that a number of domain-specific models have been published as the result of consensus building in particular areas of discourse. Some examples include the Semantics of Business Vocabulary and Business Rules (SBVR) [71], which focuses on “documenting the semantics of business vocabularies and business rules for the exchange of business vocabularies and business rules among organizations and between software tools”; or

the International Council of Museums (ICOM) International Committee for Documentation (CIDOC) Conceptual Reference Model (CIDOC CRM) [13, 52], which “provides definitions and a formal structure for describing the implicit and explicit concepts and relationships used in cultural heritage documentation”; or the Cultural Heritage Abstract Reference Model (CHARM) [25, 51]. Models like these are highly specialised in a technical area, have been created after more or less elaborate processes of consensus building among experts in the field, are published to a wide audience for shared reference, and often are provided in a machine-readable format that may allow automated processing by computer. Therefore, and according to our discussion in previous sections, they qualify as domain ontologies. Whether actual ontological thinking has been used to construct these models is sometimes difficult to say, either because this fact is not captured in the published documentation or because of the blur between ontologies and conceptual modelling that we have previously described. Some of these models, however, explicitly mention the fact that they are conceived as ontologies; for example, ISO 21127:2014 [52], the standard version of CIDOC CRM, states in the Introduction that “ISO 21127 is an ontology for cultural heritage information”.

The field of software engineering itself has also been described through a domain ontology, at least partially, by e.g. [22], which is strongly based on the ISO/IEC 24744 [56] standard “Software Engineering – Metamodel for Development Methodologies”.

Having a published, shared ontology of a domain can be enormously useful in software engineering, especially in situations where a software system is to be built in a specific domain. First of all, the domain ontology provides a readily available and common vocabulary and conceptualization for the communication during requirements elicitation and analysis. Despite no empirical studies have been carried out about this as far as I know, our experience is that software developers learn about a domain much faster and make fewer mistakes when supported by a domain ontology rather than mere input and discussion with domain experts.

Secondly, the domain ontology can be used as a starting point on which to develop the system’s domain model, along the lines proposed by domain-driven design (DDD) [17]. Usually, systems cover only a specific area of a domain, and often in a manner that is highly particular to the customer of future users; this means that, whatever ontology is taken as a base, it will likely have to be “pruned” and refined. The degree to which domain ontologies support extension and tailoring is highly variable, this being a factor with a large impact on the applicability of a domain ontology to the practice of software engineering (see below). Some kinds of systems go one step beyond and, instead of being based on a particular ontology, assume that there will be an ontology serving as conceptual basis for the processes that take place inside, but that this ontology is not fixed. These systems model the concept of ontology as part of the system’s conceptual model; it is the case, for example, of agent-based systems developed by using the FAME Agent-Oriented Modelling Language (FAML) [8]. In FAML, “Ontology” is a language primitive which, together with others such as “Agent” or “Role”, allows the system developer to organize a community of agents that exchange information in terms of an ontology, but leave the specific contents of the ontology open to be dynamically evolvable at run time. In other words, under FAML, ontologies are not constructed in design-time and then used in run-time; rather, they are constructed, used and even dynamically re-constructed at run-time.

Thirdly, the domain ontology can be used as a reference model for the interchange of information between systems. Even if the system is not built according to the ontology, it may be designed so that it can import and/or export data that conforms to it, thus enhancing its interoperability. Some domain ontologies, in fact, are heavily oriented towards this, such as CIDOC CRM, which is described in [13] as intended to “provide the ‘semantic glue’ needed to mediate between different sources of cultural heritage information”.

Some remarks are worth about the extension and tailoring of domain ontologies. Although the knowledge captured by a domain ontology is supposed to be shared, it sometimes happens that certain users of the ontology wish to alter specific aspects to suit their particular views on reality, accommodate technical constraints, or simply add detail to an abstract conceptualization. As we mentioned above, the degree to which different ontologies cater for extension and tailoring varies greatly. Some, such as CHARM [50], are explicitly conceived as abstract reference models, and *must* be extended before they are used through a series of well documented extension guidelines [49]. The fact that these ontologies are expressed in an explicit and documented language contributes to the ease of extension, since formal support makes it easier to establish the extension rules and validate whether an ontology is a true extension of the base one or simply a different ontology. As a counterexample, CIDOC CRM [52] is expressed in a language that is not named, described or documented, which makes extension difficult and, what is worse, makes it impossible to verify whether a CIDOC CRM-looking ontology is a true extension of the standard or not.

3.3 For Standardisation of Software Engineering

The third and last area of use of ontologies in software engineering is concerned with the field of software engineering itself. Practice in this field is varied and colourful, including approaches that range from the very rigorous of formal methods and high-ceremony methodologies to the hacker ethics of some agile approaches and “extreme” styles. At different points along this spectrum, different standardisation organizations have been working to produce guidelines and recommendations that may help the community to improve the ways in which we develop software systems. A good example is the SWEBOK ontology [65], based on the Software Engineering Body of Knowledge (SWEBOK), initially developed by the IEEE Computer Society and then made into an international standard as ISO/IEC TR 19759:2005 [57]. Another interesting case is that of the International Organization for Standardization (ISO), Joint Technical Committee 1, Sub-Committee 7 (JTC1/SC7), named “Software and systems engineering”. This subcommittee has been working since 1987 in the “standardization of processes, supporting tools and supporting technologies for the engineering of software products and systems”. ISO JTC1/SC7 has produced a number of standards in the areas of process lifecycles, process assessment, system architecture, open distributed processing, methodologies, testing or user documentation. Unfortunately, different standards, especially when coming from different working groups, tend to use a different conceptualization of the software engineering field, very often overlapping but incompatible [40]. For example, ISO/IEC 12207 “Software life cycle processes” [54] and ISO/IEC 15504

“Process assessment” [53] use substantially different conceptualizations of what a software process is; this is remarkable, given the fact that 15504 is supposed to establish a manner in which processes such as those defined by 12207 are to be assessed. In some cases, even standards coming out of the same working group present noticeable differences in their conceptualization; it is the case, for example, of ISO/IEC 15288 “System life cycle processes” [55] and the previously mentioned ISO/IEC 12207, which present very different views on how processes are organized and composed of smaller units. These discrepancies between standards make interoperation and communication very difficult.

To mitigate this, and after the problem had been identified and described by several key actors [40, 63], ISO JTC1/SC7 initiated a study group in 2012 with the aim to “evaluate the feasibility of preparing an ontology (a conceptual model) of the domains of interest of SC7 and its standards”. After some exploratory work, this group proposed that the major challenge to be tackled was to provide a solution to the ongoing tension between standardisation and customisation. In other words:

- standards already exist and are being actively applied by industry, so they should not be changed arbitrarily;
- at the same time, reconciling differences necessarily means that somehow standards must change.

The proposed solution was based on the idea of the gradual refinement of models, already employed for CHARM [24]. According to this idea, a definitional elements ontology (DEO) would be created to work as a very abstract representation of all the SC7 concerns and concepts. The DEO would be so abstract that it could not be applied straight away; it would need to be refined into a configured definitional ontology (CDO) whenever is needed through a set of well-defined mechanisms, such as removal of unwanted areas or extension with new concepts [42]. CDOs can be also “chained” an arbitrary number of levels by further refining a CDO into a more concrete CDO, in order to add detail in a piecemeal fashion, often to match the organizational and operational needs of the community [45]. For example, a CDO could be created from SC7’s DEO for each of the major scope areas in which SC7 works; from these first-level CDOs, each working group could derive its own particular CDO, and even a more specific CDO could be constructed for each family of standards when needed. Finally, a standard domain ontology (SDO) is an instance of a CDO that suits the needs of a particular standard, providing its conceptual foundation.

The study group proposed a proof-of-concept DEO to SC7 in late 2014, consisting of 26 classes plus associations, which are strongly based on ISO/IEC 24744 [56] and related work.

4 Outlook

In the previous sections I have described ontologies and ontological thinking from the perspective of software engineering, and in particular in relation to modelling and meta-modelling. Although ontologies have been introduced in the software engineering field

for some time now, and are being effectively used for some purposes, there are still a number of areas where much work is to be done. The hybridisation of the two fields (ontologies and software engineering, see Sect. 1) also poses new challenges. This is particularly so in the area of philosophical grounding (Sect. 3.1), where specific aspects of upper ontologies are being re-examined and questioned in recent works, such as those about physical vs. logical modelling [16], alternative modelling primitives [37], or the notion of identity [42]. This is a complex and difficult area of research where very few studies exist with a strong empirical or logical backing, and for this reason more advances are to be expected in the near future.

In the domain modelling and standardisation areas (Sects. 3.2 and 3.3), in turn, the major challenge resides in finding a suitable manner to alleviate the tension between the need for standardisation and that for customisation. The proposal from the ISO JTC1/SC7 study group, described in Sect. 3.3, is being tested in the field and will hopefully produce results in the next few years. Other approaches may also be proposed. Also in this area and connected to the previous, a significant challenge is that of consensus building. Since an ontology working as shared domain model, especially if it is to be a standard, is supposed to be accepted by a large community, agreement must be reached on what this model contains and how it represents reality. Although this is primarily a social rather than technical issue, ontology and modelling technologies must be developed so that they can accommodate the incremental construction of models and exploratory developments as required by this situation.

In conclusion, ontologies have contributed very valuable insights to the theory and practice of software engineering, especially in the subfield of conceptual modelling. But they have also created a new area of inquiry, bringing up new questions and old problems that will take long to settle.

Acknowledgements. Thank you to Brian Henderson-Sellers for the revision of a draft of this work and for his contributions to the ideas presented here.

References

1. Ackoff, R.L.: From data to wisdom. *J. Appl. Syst. Anal.* **16**, 3–9 (1989)
2. Ashburner, M., Ball, C.A., Blake, J.A., Botstein, D., Butler, H., Cherry, J.M., Davis, A.P., Dolinski, K., Dwight, S.S., Eppig, J.T., Harris, M.A., Hill, D.P., Issel-Tarver, L., Kasarskis, A., Lewis, S., Matese, J.C., Richardson, J.E., Ringwald, M., Rubin, G.M., Sherlock, G.: Gene ontology: tool for the unification of biology. *Nat. Genet.* **25**, 25–29 (2000)
3. Atkinson, C.: Supporting and applying the UML conceptual framework. In: Bézivin, J., Muller, P.-A. (eds.) *UML 1998. LNCS*, vol. 1618, pp. 21–36. Springer, Heidelberg (1999). doi:[10.1007/978-3-540-48480-6_3](https://doi.org/10.1007/978-3-540-48480-6_3)
4. Atkinson, C., Gutheil, M., Kiko, K.: On the relationship of ontologies and models. In: *Proceedings of the 2nd International Workshop on Meta-Modelling (WoMM)*. LNI 96, Karlsruhe, Germany, pp. 47–60 (2006)
5. Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. *ACM Trans. Model. Comput. Simul.* **12**(4), 290–321 (2002)
6. Atkinson, C., Kühne, T.: Model-driven development: a metamodeling foundation. *IEEE Softw.* **20**(5), 36–41 (2003)

7. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. *Sci. Am.* **284**, 29–37 (2001)
8. Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J.J., Pavon, J., Gonzalez-Perez, C.: FAML: a generic metamodel for MAS development. *IEEE Trans. Softw. Eng.* **35**(6), 841–863 (2009)
9. Bézivin, J.: On the unification power of models. *Softw. Syst. Model.* **4**(2), 171–188 (2005)
10. Bunge, M.: *Treatise on Basic Philosophy - Ontology I: The Furniture of the World*, vol. 3. Reidel, Boston (1977)
11. Bunge, M.: *Treatise on Basic Philosophy - Ontology II: A World of Systems*, vol. 4. Reidel, Boston (1979)
12. Castel, F.: Ontological computing. *Commun. ACM* **45**(2), 29–30 (2002)
13. CIDOC. The CIDOC Conceptual Reference Model (web site) (2011). <http://www.cidoc-crm.org/>. Accessed 26 Nov 2012
14. Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., Jeremaes, P.: *Object-Oriented Development: The Fusion Method*. Prentice-Hall, Englewood Cliffs (1994)
15. Corcho, O., Fernández-López, M., Gómez-Pérez, A.: Ontological engineering: principles, methods, tools and languages. In: Ruiz González, F., Calero, C., Piatini, M. (eds.) *Ontologies for Software Engineering and Software Technology*, pp. 1–48. Springer, Heidelberg (2006)
16. Eriksson, O., Henderson-Sellers, B., Ågerfalk, P.J.: Ontological and linguistic metamodeling revisited: a language use approach. *Inf. Softw. Technol.* **55**(12), 2099–2124 (2013)
17. Evans, E.: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, Boston (2003)
18. Favre, J.-M.: Foundations of meta-pyramids: languages vs. metamodels - Episode II: story of thotus the baboon. In: Bézivin, J., Heckel, R. (eds.) *Language Engineering for Model-Driven Software Development*, Dagstuhl Seminar Proceedings, 04101. IBFI, Dagstuhl (2005)
19. Gašević, D., Kaviani, N., Hatala, M.: On metamodeling in megamodels. In: Engels, G., Opdyke, B., Schmidt, Douglas C., Weil, F. (eds.) *MODELS 2007*. LNCS, vol. 4735, pp. 91–105. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-75209-7_7](https://doi.org/10.1007/978-3-540-75209-7_7)
20. Gonzalez-Perez, C.: A conceptual modelling language for the humanities and social sciences. In: Rolland, C., Castro, J., Pastor, O. (eds.) *Sixth International Conference on Research Challenges in Information Science (RCIS)*, pp. 396–401. IEEE Computer Society (2012)
21. Gonzalez-Perez, C., Henderson-Sellers, B.: A representation-theoretical analysis of the OMG modelling suite. In: *The 4th International Conference on Software Methodologies, Tools and Techniques*, 28–30 September 2005. *Frontiers in Artificial Intelligence and Applications* 129. IOS Press, Amsterdam, pp. 252–262 (2005)
22. Gonzalez-Perez, C., Henderson-Sellers, B.: An ontology for software development methodologies and endeavours. In: Ruiz González, F., Calero, C., Piatini, M. (eds.) *Ontologies for Software Engineering and Software Technology*, pp. 123–151. Springer, Heidelberg (2006)
23. Gonzalez-Perez, C., Henderson-Sellers, B.: Modelling software development methodologies: a conceptual foundation. *J. Syst. Softw.* **80**(11), 1778–1796 (2007)
24. Gonzalez-Perez, C., Martín-Rodilla, P.: Integration of archaeological datasets through the gradual refinement of models. In: Gilgny, F., et al. (eds.) *21st Century Archaeology: Concepts, Methods and Tools - Proceedings of the 42nd Annual Conference on Computer Applications and Quantitative Methods in Archaeology*, pp. 193–204. Archaeopress (2015)
25. Gonzalez-Perez, C., Parcero Oubiña, C.: A conceptual model for cultural heritage definition and motivation. In: Zhou, M., et al. (eds.) *Revive the Past: Proceeding of the 39th Conference on Computer Applications and Quantitative Methods in Archaeology*, pp. 234–244. Amsterdam University Press (2011)

26. Graham, I., Henderson-Sellers, B., Younessi, H.: The OPEN Process Specification. The OPEN Series. Harlow. Addison-Wesley Longman, Essex (UK) (1997)
27. Gregor, S.: The Nature Of Theory In Information Systems. *MIS Q.* **30**(3), 611–642 (2006)
28. Gruber, T.: A translation approach to portable ontology specifications. *Knowl. Acquisition* **5**(2), 199–220 (1993)
29. Gruber, T.: Toward principles for the design of ontologies used for knowledge sharing? *Int. J. Hum Comput Stud.* **43**(5–6), 907–928 (1995)
30. Gruber, T.: Ontology. In: Liu, L., Özsu, M.T. (eds.) *Encyclopedia of Database Systems*. Springer, New York (2009)
31. Guarino, N.: Formal ontology, conceptual analysis and knowledge representation. *Int. J. Hum Comput Stud.* **43**(5–6), 625–640 (1995)
32. Guarino, N.: Some ontological principles for designing upper level lexical resources. In: Rubio, A., et al. (eds.) *Proceedings of First International Conference on Language Resources and Evaluation*, Granada (1998)
33. Guha, R.V., Lenat, D.B.: Cyc: a midterm report. In: Buchanan, B.G., Wilkins, D.C. (eds.) *Readings in Knowledge Acquisition and Learning*, pp. 839–866. Morgan Kaufmann, New York (1993)
34. Guizzardi, G.: *Ontological Foundations for Structural Conceptual Models*. University of Twente, The Netherlands (2005)
35. Guizzardi, G., Wagner, G.: On the ontological foundations of agent concepts. In: Grundspenkis, J., Kirikova, M. (eds.) *CAiSE 2004 Workshops in Connection with The 16th Conference on Advanced Information Systems Engineering*, pp. 265–279. Riga Technical University (2004)
36. Guizzardi, G., Wagner, G.: A unified foundational ontology and some applications of it in business modeling. In: Missikoff, M. (ed.) *Enterprise Modelling and Ontologies for Interoperability*, CEUR Workshop Proceedings, vol. 125. CEUR-WS.org (2004)
37. Guizzardi, G., Zamborlini, V.: Using a trope-based foundational ontology for bridging different areas of concern in ontology-driven conceptual modeling. *Sci. Comput. Program.* **86**, 417–443 (2014)
38. Heller, B., Herre, H.: Ontological categories in GOL. *Axiomathes* **14**(1), 57–76 (2004)
39. Henderson-Sellers, B.: Bridging metamodels and ontologies in software engineering. *J. Syst. Softw.* **84**(2), 301–313 (2011)
40. Henderson-Sellers, B.: Standards harmonization: theory and practice. *Softw. Syst. Model.* **11**(2), 153–161 (2012)
41. Henderson-Sellers, B., Barbier, F.: What is this thing called aggregation? In: *TOOLS 29*, May 1999. IEEE Computer Society (1999)
42. Henderson-Sellers, B., Eriksson, O., Ågerfalk, P.J.: On the need for identity in ontology-based conceptual modelling. In: Saeki, M., Kohler, H. (eds.) *Proceedings of 11th Asia-Pacific Conference on Conceptual Modelling (APCCM 2015)*, CRPIT, Sydney, Australia, pp. 9–20 (2015)
43. Henderson-Sellers, B., Gonzalez-Perez, C.: Multi-level meta-modelling to underpin the abstract and concrete syntax for domain specific modelling languages. In: Reinhartz-Berger, I., et al. (eds.) *Domain Engineering: Product Lines, Conceptual Models, and Languages*, pp. 291–316. Springer, Heidelberg (2013)
44. Henderson-Sellers, B., Gonzalez-Perez, C., Eriksson, O., Ågerfalk, P.J., Walkerden, G.: Software modelling languages: a wish list. In: Gray, J., et al. (eds.) *IEEE/ACM 7th International Workshop on Modeling in Software Engineering (MiSE)*. IEEE Computer Society (2015)

45. Henderson-Sellers, B., Gonzalez-Perez, C., McBride, T., Low, G.: An ontology for ISO software engineering standards: 1) Creating the infrastructure. *Comput. Stand. Interfaces* **36**(3), 563–576 (2014)
46. Henderson-Sellers, B., Gonzalez-Perez, C., Walkerden, G.: An application of philosophy in software modelling and future information systems development. In: Franch, X., Soffer, P. (eds.) *CAiSE 2013. LNBIP*, vol. 148, pp. 329–340. Springer, Heidelberg (2013). doi: [10.1007/978-3-642-38490-5_31](https://doi.org/10.1007/978-3-642-38490-5_31)
47. Hesse, W.: From conceptual models to ontologies. In: Delcambre, L., Kaschek, R.H., Mayr, H.C. (eds.) *Dagstuhl Seminar on The Evolution of Conceptual Modeling*. Schloss Dagstuhl, Dagstuhl (2008)
48. Incipit. ConML Technical Specification. Incipit, CSIC (2016). http://www.conml.org/Resources_TechSpec.aspx
49. Incipit. CHARM Extension Guidelines. Incipit, CSIC (2016). <http://www.charminfo.org/Resources/Technical.aspx>
50. Incipit. CHARM Web Site (web site) (2016). <http://www.charminfo.org>. Accessed 30 May 2016
51. Incipit. CHARM White Paper. Incipit, CSIC (2016). <http://www.charminfo.org/Resources/Technical.aspx>
52. ISO. Information and documentation – a reference ontology for the interchange of cultural heritage information. ISO 21127:2014 (2014)
53. ISO/IEC. Software Process Assessment - Part 1: Concepts and Vocabulary. ISO/IEC 15504-1:2004 (2004)
54. ISO/IEC. Systems and software engineering – software life cycle processes. ISO/IEC 12207:2008 (2008)
55. ISO/IEC. Systems and software engineering – system life cycle processes. ISO/IEC 15288:2008 (2008)
56. ISO/IEC. Software Engineering - Metamodel for Development Methodologies. ISO/IEC 24744:2004 (2014). http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=62644
57. ISO/IEC. Software Engineering - Guide to the software engineering body of knowledge (SWEBOK). ISO/IEC TR 19759 (2015). http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=67604
58. Katifori, A., Halatsis, C., Lepouras, G., Vassilakis, C., Giannopoulou, E.: Ontology visualization methods — a survey. *ACM Comput. Surv.* **39**(4), 10 (2007)
59. Kühne, T.: Clarifying matters of (meta-) modeling: an author's reply. *Softw. Syst. Model.* **5**(4), 395–401 (2006)
60. Kühne, T.: Matters of (meta-) modeling. *Softw. Syst. Model.* **5**(4), 369–385 (2006)
61. Martin, J., Odell, J.: *Object-Oriented Analysis and Design*. Prentice-Hall, Englewood Cliffs (1992)
62. Masolo, C., Borgo, S., Gangemi, A., Guarino, N., Oltramari, A.: *Ontology Library*. Laboratory For Applied Ontology - ISTC-CNR (2003). <http://www.loa.istc.cnr.it/old/Papers/D18.pdf>
63. McBride, T., Henderson-Sellers, B.: The Growing Need for Alignment, N5507. ISO/IEC JTC1 SC7 (2012)
64. Mellor, S.J., Balcer, M.: *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley, Boston (2002)
65. Mendes, O., Abran, A.: Software engineering ontology: a development methodology. *Metrics News.* **9**, 68–76 (2004)
66. Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D., Miller, K.: WordNet: an on-line lexical database. *Int. J. Lexicogr.* **3**, 235–244 (1990)

67. Newell, A.: The knowledge level. *Artif. Intell.* **18**(1), 87–127 (1982)
68. OMG. MDA Guide, omg/2003-06-01. Object Management Group (2003)
69. OMG. Unified Modelling Language Specification: Infrastructure. formal/05-07-05 (2006)
70. OMG. OMG Meta Object Facility (MOF) Core Specification. formal/2013-06-01 (2013). <http://www.omg.org>
71. OMG. Semantics of Business Vocabulary and Business Rules (SBVR). formal/2015-05-07 (2015). <http://www.omg.org/spec/SBVR/>
72. Opdahl, A.L., Henderson-Sellers, B.: Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Softw. Syst. Model.* **1**(1), 43–67 (2002)
73. Opdahl, A.L., Henderson-Sellers, B., Barbier, F.: Erratum to “ontological analysis of whole-part relationships in OO models”. *Inf. Softw. Technol.* **43**(9), 577 (2001)
74. Opdahl, A.L., Henderson-Sellers, B., Barbier, F.: Ontological analysis of whole-part relationships in OO models. *Inf. Softw. Technol.* **43**(6), 387–399 (2001)
75. Partridge, C.: A Couple of Meta-ontological Choices for Ontological Architectures. LADSEB-CNR, Padova (2002)
76. Partridge, C.: Business Objects: Re-Engineering for Re-Use. 2nd edn. The BORO Centre, 412 p. (2005)
77. Partridge, C., Gonzalez-Perez, C., Henderson-Sellers, B.: Are conceptual models concept models? In: Ng, W., Storey, Veda C., Trujillo, Juan C. (eds.) *ER 2013. LNCS*, vol. 8217, pp. 96–105. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-41924-9_9](https://doi.org/10.1007/978-3-642-41924-9_9)
78. Rosemann, M., Green, P.: Developing a meta model for the Bunge-Wand-Weber ontological constructs. *Inf. Syst.* **27**(2), 75–91 (2002)
79. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.: *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs (1991)
80. Schneider, L.: How to build a foundational ontology. In: Günter, A., Kruse, R., Neumann, B. (eds.) *KI 2003. LNCS*, vol. 2821, pp. 120–134. Springer, Heidelberg (2003). doi:[10.1007/978-3-540-39451-8_10](https://doi.org/10.1007/978-3-540-39451-8_10)
81. Selic, B.: The pragmatics of model-driven development. *IEEE Softw.* **20**(5), 19–25 (2003)
82. Simons, P., Cameron, R.: A short glossary of metaphysics. In: Le Poidevin, R., et al. (eds.) *Routledge Companion to Metaphysics*, pp. 578–599. Routledge, London (2009)
83. Stanford University. Stanford Encyclopedia of Philosophy (2015). <http://plato.stanford.edu/>. Accessed 23 July 2015
84. Uschold, M.: Where are the semantics in the semantic web? *AI Mag.* **24**(3), 25–36 (2003)
85. World Wide Web Consortium. SKOS Simple Knowledge Organization System Primer (2009). <http://www.w3.org/TR/2009/NOTE-skos-primer-20090818/>
86. World Wide Web Consortium. OWL 2 Web Ontology Language (2012). <http://www.w3.org/TR/2012/REC-owl2-overview-20121211/>

Grand Timely Topics in Software Engineering
International Summer School GTTSE 2015, Braga,
Portugal, August 23-29, 2015, Tutorial Lectures
Cunha, J.; Fernandes, J.P.; Lämmel, R.; Saraiva, J.;
Zaytsev, V. (Eds.)
2017, XI, 235 p. 44 illus., Softcover
ISBN: 978-3-319-60073-4