Ontological and linguistic metamodelling revisited: 
A language use approach

Owen Eriksson\textsuperscript{a,}\textsuperscript{*}, Brian Henderson-Sellers\textsuperscript{b}, Pär J. Ågerfalk\textsuperscript{a}

\textsuperscript{a}Department of Informatics and Media, Uppsala University, Box 513, 751 20 UPPSALA, Sweden
\textsuperscript{b}School of Software, University of Technology, PO Box 123, Broadway NSW 2007, Sydney, Australia

\textbf{A R T I C L E   I N F O}

Article history:
Received 4 July 2012
Received in revised form 31 May 2013
Accepted 19 July 2013
Available online 9 August 2013

Keywords:
Concepts
Speech act theory
Set theory
Metamodel

\textbf{A B S T R A C T}

\textbf{Context}: Although metamodelling is generally accepted as important for our understanding of software and systems development, arguments about the validity and utility of ontological versus linguistic metamodelling continue.

\textbf{Objective}: The paper examines the traditional, metamodel-focused construction of modelling languages in the context of language use, and particularly speech act theory. These concepts are then applied to the problems introduced by the "Orthogonal Classification Architecture" that is often called the ontological/linguistic paradox. The aim of the paper is to show how it is possible to overcome these problems.

\textbf{Method}: The paper adopts a conceptual-analytical approach by revisiting the published arguments and developing an alternative metamodelling architecture based on language use.

\textbf{Results}: The analysis shows that when we apply a language use perspective of meaning to traditional modelling concepts, a number of incongruities and misconceptions in the traditional approaches are revealed – issues that are not evident in previous work based primarily on set theory. Clearly differentiating between the extensional and intensional aspects of class concepts (as sets) and also between objects (in the social world) and things (in the physical world) allows for a deeper understanding to be gained of the relationship between the ontological and linguistic views promulgated in the modelling world.

\textbf{Conclusions}: We propose that a viewpoint that integrates language use ideas into traditional modelling (and metamodelling) is vital, and stress that meaning is not inherent in the physical world; meaning, and thus socially valid objects, are constructed by use of language, which may or may not establish a one-to-one correspondence relationship between objects and physical things.

\textcopyright{} 2013 Elsevier B.V. All rights reserved.

\section{1. Introduction}

Conceptual modelling is used in information systems and software development to enable conceptual understanding of the systems, the information they contain, and the processes by which they come about. Conceptual models facilitate the understanding of information systems requirements and, more generally, enhance the quality of information systems development. Conceptual modelling is a term commonly used to indicate modelling in the software context that is independent of the constraints of programming languages. It focuses on the description of real-world (business-focused) problems and how to represent them in models and could therefore be loosely related to requirements engineering, business systems analysis, and enterprise engineering.

Modelling uses a modelling language to communicate information about the models, be they for system design or for processes and methodologies. A modelling language consists of, \textit{inter alia}, an abstract syntax, a concrete syntax (notation)\textsuperscript{1} and semantics. To ensure quality and consistency, modelling languages need to be clearly defined so that their use is consistent across development teams, countries, etc. Much of the work in modelling languages for computing contexts (software engineering, information systems) over the last several decades has been focused on ‘general purpose modelling languages’, such as the now standard Unified Modeling Language\textsuperscript{TM}, UML\textsuperscript{®} [1,2]. More recently, domain-specific modelling languages (DSMLs) have been intensively researched and developed to practical solutions [3]. Here, we focus on general purpose modelling languages and eschew discussions about DSMLs.

Although there are several ways of writing down formal definitions of modelling languages, one frequently used is that of the metamodel, defined as ‘a model of models’, which defines the abstract syntax, often itself expressed using UML’s notation (typically a class diagram) together with additional behavioural and semantic constraints (perhaps using OCL or a similar logic-based

\footnote{1 Although an important element, we do not discuss the notational aspects of a modelling language in any detail in this paper.}
Metamodels focus on the concepts and rules of the methods and models used in information systems development and software engineering [4]. Metamodelling, i.e. how to create metamodels, is an important research area because there is a need for shared conceptualisations in software and systems engineering in order to avoid misunderstandings and barriers of communication in the development of IT systems [5]. Consequently, the application of metamodels is increasingly recommended. The use of metamodels for conceptual modelling, especially in the context of software engineering, has a relatively short history, in comparison to its much longer use in database modelling. The (early, 1991) CDIF standard was based on a three-level hierarchy (as summarised more recently in [6]). A few years later, suggestions by Carmichael [7] and Henderson-Sellers [8] led to the proposal to use this approach to bring together the plethora of object-oriented modelling notations then extant [9,10]. The first standard to utilise this idea of metamodelling for object technology standards was the UML [1], standardised by the Object Management Group (OMG), which introduced the four-level hierarchy of Fig. 1, with an expressed aim of providing an infrastructure “to support the creation, manipulation and interchange of meta models” [11]. It provides a metametamodel at the top layer, called the M3 layer (or MOF: Meta-Object Facility). An M3-model defines the language that is used for building metamodels, which are understood to define modelling languages at the M2 layer. The most prominent example of a Layer 2 metamodel is the model that defines the UML. These M2-models describe elements of the M1-layer, which are thus models written in UML. The elements in these M1 models each represent concepts pertinent to the real-world (or computer world) domain under investigation. Individuals classified as forming to a concept are often called ‘instances’. An instance is seen to be atomic, as compared to concepts and classes that can be represented mathematically by sets or categories e.g. [12]. (Other multilevel constructions are well presented in [13], wherein the linking of language-based and model-based approaches is similar to that in [14].) However, in the end (p. 290), the author [13] eschews a language use focus in favour of the (to-date) traditional structural (a.k.a. linguistic) metamodelling framework, as employed by the OMG.

“The primary responsibility of the metamodel (M2) layer is to define a language for specifying models” whereas the purpose of the M1 layer is said to be to define a language that describes an information domain [1, pp. 7–8]. Thus, Class belongs to the M2 layer and Horse (an instance of Class) is part of the M1 domain language. The last layer is the M0-layer, said to describe run-time instances [15, p. 17] or objects of the model – thus emphasising UML’s focus on software objects rather than objects in the real world. Notwithstanding, some authors interpret ‘M0’ as being the ‘real world’ – as seen in some diagrams abstracted from extant literature.

In the first versions of UML, there was an M2 class called Object, which led to confusion as to whether instances of this class existed at level M1 or at the more natural M0 [16]. Consequently, in Version 2, the M2 Object class was replaced by InstanceSpecification. This allows for instances of InstanceSpecification to be created as part of the M1 model, despite referring to an M1 entity. The UML documentation [15] thus notes that the instances at the M0 level should not be confused with an instance of an InstanceSpecification, which is used simply as an illustration (a snapshot) of the class, e.g. \texttt{Prancer:Horse} (or the more anonymous: \texttt{Horse}) at the

![Fig. 1. Four-layer hierarchy of the Object Management Group (OMG).](image-url)
M1 level is not an instance of the class (type) Horse – according to the OMG documentation of the UML. This is obviously confusing, and the problem occurs because of the use of ‘strict metamodelling’, which is the basic principle in the linguistic metamodelling utilised by the OMG. This principle states that the <<instanceOf>> relationship is always between adjacent meta layers, and that it cannot exist within any single layer. The ‘strict metamodelling’ principle also states that if a model A is an instance of another model B then every element of A must be an instance of some element in B [17]. Thus the <<instanceOf>> relationship is a type-instance relationship between a model element at the type level (classifier) and its instances. However, it is unclear how the objects at the M0 relate to the world. According to [15, p. 16] “The things that are being modeled reside outside the metamodel hierarchy” but nothing is said about how they relate to real-world things. In other words, the focus of the M2 layer is said to be that of ‘language’ and that of the M1 layer to be a ‘domain language’; there is nothing said about how language relates to the world. As a consequence, confusion prevails as modellers believe, contrary to the above quotations from the OMG literature, that M0 model elements are real world entities rather than models (or run-time instances of model elements) of the real world entities (see Fig. 1 above which is an adapted version of [15, Fig. 7.8]. We will later show that the relationship between the model ‘objects’ at the M1 level and the real world things is not that of type-instance but is a correspondence relationship (see also later discussion of Fig. 12).

Fig. 2. Typical framework proposed in the literature to explain ontological versus linguistic metamodelling (slightly modified from [22]).

Another modelling domain in which instanceOf relationships between layers have been adopted is that of domain ontologies. Domain ontologies are frameworks for organising information and are used in, for example, the Semantic Web, systems engineering, software engineering, information architecture and information standardisation as a form of representation about the world or some part of it. Domain ontology engineering aims to make explicit the knowledge contained within a specific area of interest, typically in the real world e.g. newspaper industry, wine industry. Establishing such a domain ontology both advises and constrains the modeller to using only concepts of the domain ontology in their (conceptual) model.

The creation of domain ontologies is often labelled ontological metamodelling in software engineering [18]. In a nutshell, Fig. 2 shows a typical example of so-called ontological metamodelling in which a class in a (say UML) model (here Shetland Pony) is argued to be an instance of another class (here Breed) that is therefore ‘meta’ to the first class (Shetland Pony) but, paradoxically, cannot be considered to be part of the underpinning metamodel (right hand side of Fig. 2) [18]. In a number of papers, possible resolutions of the so-called ‘linguistic versus ontological metamodelling’ debate [4] are explored. For example, Atkinson and Kühne [19] analyse various multi-dimensional representations that combine ontological and linguistic modelling styles into one diagram – an example, called by them the ‘Orthogonal Classification Architecture’ [20], is shown in Fig. 2, later developed into the Level-agnostic Modelling Language (LML) [21].

Fig. 2 also perpetuates the ambiguity problem noted above regarding whether the symbol labelled ‘Object’ is an M2 class that is an ontological instance-of Class (which it must be if both Object and Class are in the same linguistic level) or whether ‘objects’ (perhaps by definition) belong to level M1 (as depicted in Fig. 1). In the latter case, we can take the Object–Class–Metaclass chain as illustrating the core of the M0–M1–M2 chain of Fig. 1. Thus, considering the problem posed by Fig. 2 in more detail, we could interpret the Prancer–Shetland Pony–Breed chain of instanceOf relationships as paralleling this M0–M1–M2 chain. Thus, if the latter three concepts denote three different metamodel levels (M0, M1, M2 in the OMG hierarchy of Fig. 1) then, it is argued by several authors, surely concepts denote three different metamodel levels (M0, M1, M2 in the OMG hierarchy of Fig. 1) then, it is argued by several authors, surely the former chain can be interpreted in the same way, i.e. the Breed class is a metatype with respect to the Shetland Pony class. But clearly such a Breed metatype could not be expected to be part of the M2 layer defining the UML, i.e. one would not expect nor wish to find a Breed class alongside a Class class in an M2-level metamodel of a general purpose modelling language like the UML. This wish to create a parallelism between the O0–O1–O2 chain of Fig. 2 and the M0–M1–M2 chain of Fig. 1 in which Breed (O2) is somehow regarded as ‘meta’ to the Shetland Pony class creates a paradox since clearly Breed can never be a metaclass in a MOF-based modelling language (such as UML).

Since contemporary metamodelling uses a type-instance pattern as its core technique, a further serious theoretical problem can be identified with the architecture of Fig. 1 [14]. Accepting that multi-level modelling involves a ‘chain’ of two (or more) type models introduces a serious concern, since this is contradictory to the set-based rules of modelling by which a type model links a type with objects conformant to that type – a relationship that is only valid over a single pair of contiguous metatypes (Fig. 3). An instance e.g. Horse as an instance of Class in Fig. 3 cannot be further instantiated since it is already an individual (an object – an object is an instance of a class in object-oriented modelling). In other words, a concept cannot be an instance of another concept – an assertion also strongly supported by theories of language use, such as speech act theory [23] (see also later discussion in Sections 4 and 5). What appears to occur in practice is that developers insert an isotypical mapping from this instance (called obj1:Class in the right hand side of Fig. 3) to a Horse class in the model (M1 domain) [14]. Since Horse is now a class, according to this argument it can be instantiated and create a number of objects of type Horse at the M0 level e.g. Prancer:Horse, Dobbin:Horse.

However, this is argued [14] to be ‘only a trick’ that is ‘far from acceptable as part of the foundational ideas of an engineering discipline.’ There are two arguments proposed in [14]: (1) ‘The real
classes are not derived from UML but from the objects derived from UML by developers that use their subjective judgment to synthesise them as necessary; and (2) ‘modelling tools implemented as software systems (as most are) need to implement a fully formalised modelling infrastructure in order to support the necessary functionality. In our experience as both modellers and modelling tool implementers, using instance-of relationships only is not enough to achieve this, and additional workarounds must be added to traverse the “hidden” isotypical mapping’ – as described in [14].

The problems described above can be related to the nature of meaning, language use, and the relationship between language and world, which are the main topics in speech-act theory, one of the core theories in the philosophy of language. In this theory, the focus is on analysing what speakers and listeners do with language in communication and a distinction is made between sentences and speech acts (utterances). However, they are dependent on each other if we want to understand the notion of meaning. For example, the sentence “Prancer is a horse” can be used to perform a speech act in a specific context and then (1) the pragmatic meaning of the sentence is established by the performance and interpretation of a speech act in that context. This means that the same sentence, when uttered, could have a different meaning in different contexts. However, the speech act performed is also dependent on (2) the linguistic meaning (semantic or standard meaning) of the sentence used [24, p. 69]. We claim that the problems described above arise because the domain language at the M1 level has not been analysed in terms of speech acts used in a context, but only as a formal sentence detached from its actual use. To quote Searle [23, p. 17] “It would be as if baseball was only studied as a formal system of rules without the game”. This means that if we analyse the M1 level based on speech-act theory, we can better explain how a language at the M1 level is related to the world, i.e. to its problem domain, and thus resolve these problems. Furthermore, to date, the focus of modelling languages such as UML has been the description and definition of that (artificial) language by use of a metamodel at the M2 level (often with additional OCL constraints). Little attention has been paid to our understanding of natural languages and language use, and how this natural language-focused knowledge may be useful for defining and using artificial modelling languages for software modelling at the M2 level.

Language philosophers and linguists analyse natural languages and their usage in their search for underpinning rules, under which conditions a given utterance or sequences of utterances are meaningful (i.e. acceptable to all concerned) [23,25–27]. This is based on reconstructive analysis of language use. Thus, the analysis is based on utterances and the rules that people must follow in order to succeed with these speech acts and is essentially a reconstruction exercise. This analysis discloses that there are general functions in language, and that people must comply with certain general rules and conventions in order to be able to use these functions when they communicate. These functions are quite independent of national differences and contexts, despite the fact that the concepts and terms used differ, which implies that language and its underlying rules are dependent on local variations and on different contexts of use [28,29]. Thus the conventions and rules of natural language are implicit and in constant flux under the pressure of the language use in different contexts [e.g. 30]. In contrast, modelling languages, and domain ontologies are artificial and, although they too evolve, they only do so by someone explicitly and formally changing the underpinning rules – the reverse of how natural spoken languages evolve. Those modelling language rules may be embodied in a Prolog-style set of rules, in an ontology, in a metamodel, or some combination thereof [31]. Thus, the contrast between natural languages and modelling languages is that the rules used in natural language are much more implicit and are learned and changed over the years as people use them in everyday life. Modelling the requirements for an IT system, which then have to be translated into formal language systems such as programming and database languages, implies that, by using a modelling language, it must be possible to bridge the gap between informal and formal languages as well as between reality and the conceptual model that forms the basis for the design model. Indeed, a modelling language like the UML purports to span both conceptual models and design models. This implies that, during the development process, different user groups and system developers have to come to an agreement about how the rules of the M1 level language should be designed and implemented in a formal language system. This also implies that the metamodels used to define and analyse modelling languages should also be based on an understanding of natural language use, and not only on formal languages. Our approach is based on the assumption that the prob-

Fig. 3. An instance (e.g. Horse as an instance of Class) cannot be further instantiated since it is already an individual. The figure shows on the left-hand side, Horse as a class that is also an instance of Class, which is impossible within the conventional object-oriented paradigm; the asterisk signals this diagram as illegal. On the right-hand side, Horse is shown as a class that is specified by the objects obj1 through a forward-looking isotypical interpretive mapping (depicted as a solid arrow).
2. The linguistic/ontological metamodelling ‘paradox’

All modelling and metamodelling (the latter just a specific application of modelling techniques to a more abstract domain e.g. page 290 of [13]) relies on the notion of a model representing a set of entities (often called a ‘system’) by means of an abstraction mapping [34]. Linguistic metamodelling (i.e. the modelling of languages, see Fig. 1) focuses on the use of an abstraction mapping that is many-to-one, creating a type–object relationship, and one in which the language used at the type level is different from that at the instance level – this will typically be observed in the different vocabularies that are used. The M2 layer (see Fig. 1) describes the vocabulary of the modelling language, and the M1 layer describes the vocabulary of the problem domain (the domain language). In the M2 layer, we can find Class, Instance and Attribute, and in the M1 layer we can find e.g. Horse. Horse can also be found at the M0 layer in Fig. 1, representing an object or runtime instance, which is unclear, because “no explanation exists of how entities in the infrastructure relate to the real world. Does the infrastructure accommodate real-world elements or do they lie outside?” [18, p. 38]. In contrast, ontological metamodelling typically uses the same vocabulary for all ontological levels in the model: Breed at the O2 level, Shetland Pony at the O1 level and Prancer at the O0 level (see Fig. 2).

This is reminiscent of the distinction made [5] between type models and token models. Both use abstraction but only the latter use the same vocabulary for the model as for the system being modelled. In other words, token models are the result of a modelling approach in which abstraction merely leads to a change in granularity – a transformation that is one-to-one – whereas type models (our main interest here) represent a one-to-many relationship between the type and its (several) instances. This suggests that linguistic metamodelling, which relies on type–instance semantics, is a different kind of (meta)modelling than ontological modelling, which only uses granularity abstractions. In Fig. 2, for example, all the ontological levels use the vocabulary relevant to the domain of horses (i.e. the real world), whereas linguistic metamodelling links abstract concepts such as “class” to its many instances (horse, cow, sheep and so on).

A ‘problem’ arises if we wish to retain the strict metamodelling hierarchy of Fig. 1 and at the same time satisfy both ontological and linguistic concerns (as noted briefly in Section 1). This can be more clearly seen if we redraw Fig. 2 to focus on the linguistic rather than the ontological levels (Fig. 4). We note that the Metaclass said to be at linguistic level M3 has a linguistic \(<\text{instanceOf}>\) link to Breed at M1. Since \(<\text{instanceOf}>\) links are only permitted between consecutive Mx layers in strict metamodelling, a proposed instantiation relationship between M3 and M1 is invalid. One solution is to relocate the Metaclass class to level M2. However, this would make the Metaclass–Class and Metaclass–Object linguistic \(<\text{instanceOf}>\) links reside within a layer, in contradiction to the strict metamodelling rule that states that linguistic instantiation operates between layers, and only ontological \(<\text{instanceOf}>\) relationships are permitted within layers. We will return to this problem and present our own solution in Section 7.

5 There are many other possible ‘solutions’ – as summarised in [4]. See also [33,50].
3. A possible resolution – using powertypes

Powertypes were introduced to software engineering to help address the concerns of representing classes, their subtypes and their instances [35], essentially from a set-theoretic viewpoint. In particular, the use of powertypes for modelling software development methodologies provides support not only for method design but also for method enactment [36].

Although powertype patterns [37] are seen more frequently in metamodelling treatises (than in modelling), they have in fact a long history in object modelling [38]. A powertype is defined [35] as a type the instances of which are subtypes of another type (called the partitioned type – more correctly, it is a mathematical ‘partition’). The relationship between these two types is said to be that of classification [38, p. 257] or its inverse, partitioning. In modelling in software engineering, it is typically assumed that the elements, \( A_i \), of a powertype, \( P(B) \) (Fig. 5), form a complete and exhaustive partition of the superset \( B \). Furthermore, the powertype and the partitioned type are related indirectly through the entities that are instances of the powertype and, at the same time, subtypes of the partitioned type (Fig. 5), which means that these instances are, at the same time, both objects as well as classes; hence their appelation as ‘clabjects’ (= cla(ss) + (o)bject) [17], which means that the construct has a type and an object facet at the same time.

As noted in Section 1 (Fig. 3), it is untenable to have two instanceOf relationships connecting three classes (as sets) in a linear fashion. Notwithstanding this theoretical ‘error’, many examples, even those used to illustrate ontological metamodelling, appear to ignore this problem. For example, in Fig. 2, we see that Shetland pony (a class) is an ontological instance of Breed (also a class). In set theoretic terms, this can only be valid if Breed is a power set (or more strictly a family of sets – see above) (Fig. 6) such that each element of the Breed set maps to a part of a Horse set (itself a partition). The Breed set in this example has only three elements – and each Breed element supplies a name to each of the subclasses of the Horse class. Fig. 7 summarises this using class notation in which the powertype structure that is implicit in Fig. 6 is shown explicitly. It is important to notice that the models in Figs. 5–7 assume that the relationship between the partitioned type (e.g. Horse) and the powertype (e.g. Breed) is a classification relationship and that the partitioning relationship between the powertype and the partitioned type implies that elements of the Powertype (e.g. Shetland Pony) are objects with their own instances, which we will show is misleading. Although useful in certain aspects of metamodelling (e.g. [41]), the powertype, as used in software engineering today, ignores both the ontological ideas of a foundational ontology, such as the Unified Foundational Ontology [42], as well as the pragmatic framework of language use theory. These will both be explored in this paper as providing a higher quality solution than the sole use of powertypes (see Section 7).
4. The need for a foundational ontology and a pragmatic theory of meaning

Sections 2 and 3 have shown that there are no immediately satisfactory solutions to the ontological/linguistic metamodelling paradox. In Sections 5.3 and 7.1, we will present improved solutions to the ontological/linguistic metamodelling paradox, based on a language use understanding of metamodelling. However, in order to do that, we have to pave the way by describing the need for a foundational ontology (Section 4.1), a pragmatic theory of meaning (Section 4.2) and an understanding of natural language and how it is used [43–46] (Section 4.3).

4.1. Foundational ontology

Surprisingly, despite the name, i.e. ‘ontological’ metamodelling, research by software engineering and information systems ontologists [42,47–49], which focus on foundational ontologies, has not been incorporated into linguistic/ontological metamodelling discussions to date [18,33,50,51]. In philosophy, ontology is defined as [52] (1) a branch of philosophy concerned with the nature and relations of being; and (2) a particular theory about the nature of being or the kinds of things that have existence. Epistemology, on the other hand, which is a concept closely related to ontology, means the study of human-created knowledge. The majority of writers in the metamodelling literature have taken to use the term ontology to mean epistemology [53] and, despite its technical incorrectness, we will also use it in this sense. When referring to the higher-level ontology (one that contains concepts such as Sortal and Universal), we will use the term ‘foundational ontology’. The basic needs for foundational ontologies in software engineering and information systems modelling are to:

(i) promote reuse in a higher level of abstraction aimed at maximising the reuse of domain models [48];
(ii) produce domain specifications that are truthful to reality [48,54];
(iii) establish theoretical foundations for conceptual modelling languages [55].

Foundational ontologies would belong to the M2 level of the OMG four-layer hierarchy since they are theories about kinds (types) and hence give type abstractions of elements in the domain ontology (M1 level) i.e. they provide foundational concepts [42] that are commensurate with the classes in a MOF-based M2 metamodel [53]. The elements of such a foundational ontology must include concepts such as sortal (class concept), universal, set, property, attribute and proper names (identifiers) and a commitment of how and which entities exist in the world. In contrast, [18] would appear to require these meta-ontologies to also include Breed and all classes of that ilk, which is clearly undesirable for a general purpose modelling approach. Furthermore, since true instantiation (as here) must be confined within a single (linguistic) metalayer, it is clear that ALL classes in published examples, such as in Figs. 1–4, belong to the model domain (M1).

Universals constitute a basic construct in foundational ontologies. Universals are general language terms that apply to a number of individuals, Thus, Horse, Breed, Colour, being Enrolled, are all considered to be universals. Some universals can also be used as Sortals, often called “Sortal Universals”, i.e. “Class concepts” or “Class” for short. A sortal is a “concept grasp of which includes knowledge of criteria of individuation and reidentification, such as dog or concerto, but not flesh or music” [56]. These universals can thus be roughly equated to a (UML-like) concrete class, which consists of an extensional and intensional part.

The intensional part of a class (the type level or concept level) provides a principle of application [48], by which we judge whether a general term applies to an object, e.g. whether something is a Dog. The principle of application determines valid individuals in the class’s extension and includes defining predicates. So, the defining predicate for the Dog class can be “A domesticated carnivorous mammal (Canis familiaris) related to foxes and wolves” and the defining predicate for the Dog Breed class can be “groups of closely related and visibly similar domestic dogs, which are all of the subspecies Canis lupus familiaris, having characteristic traits that are selected and maintained by humans, bred from a known foundation stock” [57]. We also need a principle of identity (or identification) in order to distinguish between the objects within the class. The principle of identification is based on constitutive rules that define how the object should be instantiated and subsequently identified and referred to in different use contexts.

We need the intension to convey the meaning of the extensional part, but we need the extensional part of a class in order to make identity and quantification statements (this will be elaborated on later in Section 4.3).

Of particular relevance to this study is the distinction between Substantial and Moment classes. Substantial classes (Fig. 8) are classes where the individuals have a 1:1 correspondence to substantial things (physical things) and can reasonably be called ‘kind’. These are, for example, Horse, Person, and Dog. In contrast, a Mo-
The substantial class (Fig. 8) supplies a universal that can be associated with a Substantial Class, which means that a Moment Individual can be associated to many Substantial Individuals.

Thus, in our second example domain, Dog is a substantial class whereas Breed is not. Breed can be seen as a moment class of the Dog class, i.e., the Breed class provides a partitioning rule on the Dog class. Since each part or block of the substantial class (seen as a partition) has a particular value of the Breed class (i.e. an instance of the Breed class) associated with it, the instances Collie, Labrador and Boxer of the moment class Breed have the function of being names of parts of the substantial class (Fig. 9). As a consequence, an element of the Dog class (e.g. Fido) cannot be a substantial individual of Collie as assumed in Fig. 2, because this is not a type-instance relationship, Collie being an instance of a moment class and not a type. The relationship from Collie to Fido is a partitioning relationship between different classes not a type-instance relationship (Fig. 9). The substantial individuals can only be instances of the Dog class because this is the substantial class in this domain. The substantial class and the moment class have different defining sortal predicates; thus, each object can only be an instance of one class and a type-instance relationship can only exist between a class and its individuals. As we have seen, Fido is thus a substantial individual of the Dog class, and Collie is a moment individual of the moment class Breed. This will be explained in more detail based on a language use viewpoint in Sections 5 and 7.

It is also important to distinguish between the individual of a class (the object) and a physical thing, as we noted much earlier. Bunge makes a distinction between concrete (physical) things and conceptual objects [47]. Things have intrinsic (physical, substantial) properties, conceptual objects do not; they are characterised by attributes that humans define them to have. Such constructs are objects that are studied and created in logic, mathematics and semantics. Searle claims that an object is an instantiation of a concept (class); thus objects are conceptual language constructs with a semantic meaning [23]. Based on this idea, Eriksson and Ågerfalk coined the concept of an institutional object, which is an object associated with deontic powers [43], i.e., rights and obligations.

4.2. A pragmatic theory of meaning

In their attempts to represent aspects of the real world (in our context as a precursor to information systems development), modellers tend to assume that the real world has intrinsic meaning; that a language’s value is simply to represent that external world meaning, i.e. that their task is to describe those real-world semantics. Indeed, Laarman and Kurtev [50] observe that “metamodellers are not aware of the real world meaning of the language constructs”. It is important to realise that modellers use substantial classes such as people, cars, animals and buildings when they model. However, they are concerned primarily with the meaning people ascribe to these things [58], i.e., they have to understand and develop linguistic and social rules that form the basis for communication and social interaction in a certain context. For example, physical dogs are important to a kennel club primarily because of the purpose and pragmatic meaning ascribed to them.

Physical things (corresponding to ‘brute facts’) exist in the world, but they have no meaning until we (humanity) have words to describe them and hence provide the ability to refer to them [23,59]. Brute facts exist independently of language and concern physical (brute) things and their properties (e.g. a physical dog). They only rely on the language so that the facts can be communicated; for example, the use of the sentence “Fido is a dog.” in a speech act. However, an understanding of any such thing (a brute fact) is not useful until several people are able to share that same understanding using language (and here resides a link to domain ontologies). This shared understanding [60] provides the
basis for human communication and, ultimately, modelling. In other words, until an object exists, and thus is instantiated in language – in a way that can be agreed upon by multiple personages – it is not possible to even start discussing how to model it. The pragmatic theory of meaning sets out to explain what it means to understand something in the world that is communatively mediated by speech acts. Meaning is reaching an understanding with someone about something in the world using a linguistic expression [61]. This is the reverse of the traditional modeller’s mindset, which is therefore, we observe, not linked to language use. This paper redresses the imbalance and identifies missing elements in traditional modelling with the aim of improving our modelling capability in information systems development and software engineering.

4.3. Language use

How language is used to reach understanding and how it relates to the world is the main focus of speech act theory [23,25,27,59,60]. In speech acts, sentences are used with a communicative intent. A speech act \( R(p) \) consists of a propositional content \( p \) associated with an illocutionary force \( F \). The force \( F \) indicates the purpose of the speech act, and how the proposition should correspond to the world (the direction of fit between world and word). For example, assertives have a word to world direction of fit, because, in order to be true, the word must correspond to facts in the world e.g. the assertive “I assert that The Queen Elizabeth is a ship”. According to Searle [62], declaratives have (see below) a double direction because they create new linguistic or institutional facts. Linguistic and institutional facts require language and special human institutions for their very existence. For example, performing the successful declarative speech act “I declare this ship to be The Queen Elizabeth” is enough to bring about the match between word and world, i.e. the authorised name of a ship will be The Queen Elizabeth (an institutional fact). This implies that the propositional content of a speech act \( p \) (The Queen Elizabeth, ship), “the propositional act”, not only describes an object, it also fulfils four important functions: the classification, instantiation, referring and predicating functions. Searle [23, p. 25] claims that the standard grammatical form of performing a speech act is the whole sentence; the illocutionary force of the speech act could be explicitly expressed by using so-called performative verbs like “assert”, “command”, “promise” or “declare”. Although the propositional act must be analysed together with the illocutionary force in order to understand the pragmatic meaning of the speech act, often in actual speech situations the context will make clear which force is indicated so that the explicit verb is omitted. Since the focus in our analyses will be a propositional act, we will leave out the explicit performative verb in our examples. The standard grammatical form of performing propositional acts “are grammatical predicates for the act of predication and proper names, pronouns and certain other sorts of noun phrases for reference” [23, p. 25]. Consider two people talking to each other in a park and one of them utters the speech act:

“Fido is 50 cm tall.” In this speech act, the proper name, i.e. the identifier Fido, is used to perform the referring function and the grammatical predicate “is 50 cm tall” for the act of predication, which is used to describe an object, in this case to describe a physical property of a dog.

In order to refer successfully, the following contextual conditions must be fulfilled [23]:

- There must exist one and only one object to which the speaker’s utterance of the expression applies.
- The hearer must be given sufficient means to identify the object from the speaker’s utterance.

Imagine that, in response to the speech act uttered above, the other person says “Who or what is Fido?” The first speaker then makes this identifying statement, now using a definite description “The brown dog in the park”. Considering that there is just one brown dog in sight, the other person then says “I see; so that’s Fido”.

We use language to refer in two ways [23]:

1. By using an identifier, such as “Fido”. An identifier can also be a number or a code string.
2. By using complex noun phrases in the singular. These expressions are also called “definite descriptions”, for example, “the brown dog in the park”.

Thus, there are two types of reference mechanisms in language: the definite description and the identifier, and we can choose either of them as the principle for identification for the objects of a class. Importantly, however, an identifier is not the same as a definite description. In fact, taking a definite description of an object as the meaning of its identifier would lead to the odd consequence that the meaning of the identifier would change if there were any change at all in the object [23]. For example, if for some reason a physical property of Fido was changed so that, say, the colour attribute had to be changed to yellow, the identifier “Fido” would still refer to the same object but the definite clause “the brown dog” would not. This means that the identifier and the definite description only refer to the same object in a specific use situation. If they had exactly the same meaning and the meaning of the identifier changed depending on how the attributes changed over time, then the identifier could not represent the existence of the object over time. It would also imply that we would only be able to refer to a physical thing by describing it, which is precisely what the identifier construct avoids and what constitutes the distinction between identifiers and definite descriptions [23]. Thus, both the identifier and the definitive description can be used to refer to an object but the identifier can refer without describing, which makes it the optimal construct for isolating the referring function of language from the predicative function.

Furthermore the propositional content can also be used to instantiate objects. For example, consider the speech act “Fido is a dog”. This is an identifying statement where an identifier (Fido) and a class name (Dog) are used. It is tempting to see a proposition of that form as always assertive referring to and describing an already existing object. However, this conceals the crucial role of existential propositions for making identifying statements, because the possibility of giving an identifying statement a non-existential form like “Fido is a dog” depends on an implicit or explicit existential proposition of the form “There exists one and only one dog with the name of Fido” in a certain context of use. To quote Searle [23, p. 93], “One might say: underlying our conception of any particular object is a true uniquely existential proposition”. When identifiers are used in successful existential proposition acts (like the one above), they cannot be said to refer because “If it did the precondition of its having a truth value would guarantee its truth”. This means that existential propositional acts are not assertives: they are declaratives. “An existential statement does not refer to an object and state that it exists, rather it expresses a concept and that that concept is instantiated” [23, p. 165]. This also means that an object can only be instantiated in one class and subsequently be referred to belonging to that one class, because in order to “secure continuity of reference, we need a criterion of identity, and the general term associated with the name provides that criterion” [23, p. 167]. For example, the statement “Fido” would not qualify as an identifying statement because it does not have an implicit or explicit link to a class name (a link to a universal).
Consider these two identifying statements, referring to a cat and a dog in the same context of use where the object Fido has previously been instantiated in two separate classes naming to different living things.

“Fido is a cat”
“Fido is a dog”

The object referred to by the identifier “Fido” is used with two different principles of application and meanings. Without a link to the Dog or Cat class, we would not be able use “Fido” as an identifier because it would pick out two distinct and different objects. One explicitly or implicitly has to relate to class names in order to provide the criterion for identity. You can only use “Fido” in social contexts, where this identifier functions as a proper reference mechanism, i.e. in contexts where the name has a meaning and thus can be used to refer uniquely to an object. All identifiers, no matter if they are names, codes or numbers, are dependent on the ascribed meaning they have in the different social contexts where they are used. A major problem with traditional modelling approaches is that they assume that the identity of an object is unproblematic because it can always be based on an inherent unique property of a thing, or the very substance of a thing. These are deceptive assumptions [25].

To quote Searle [23, p. 75] “A proper name could only be a proper name if there is a genuine difference between the name and the thing named. If they were the same, the notion of naming and referring can have no application” The important conclusion using speech act theory is that identity is a truly social construct. Traditional modelling approaches assume that the identity of an object already exists in the material world, and thus does not have to be socially constructed through the use of language.

However, the same physical thing can be referred to by using several identifiers and thus correspond to many objects. The best known example of that in the literature is the “Morning star”, “Evening star” and “Venus” example presented by Frege [63], where there are three objects and identifiers that correspond to the same physical thing, but with different meanings. A thorough discussion about the correspondence relationship between things and objects in an IT-system modelling context can be found in [43] where, for example, the ISO-VIN and licence numbers of a physical vehicle are shown to be identifiers of two different vehicle objects that correspond to a single physical vehicle (a thing).

The discussion above is of interest because it reveals that:

- Things and their properties are substances that exist independently of language and carry no inherent meaning (e.g. physical dogs). They represent brute facts.
- Objects (such as Fido) have to be instantiated in language because they have to be mutually understood. Thus, they are language constructs, which means that objects and things are not the same.
- Class names are used for the classification function of language and an object can only be instantiated and subsequently referred to using one class because the intension (the type) provides both the principle of application and identification i.e., the criteria for which objects could be instantiated in the class.
- Identifiers are used together with class names in existential propositions, which are declaratives, in order to perform the instantiation function of language.
- Identifiers are used for the referring function of language when they are used in a non-existential proposition.
- Attributes are language constructs used for describing objects in performing the predicating function of language.

5. The Breed–Collie–Fido relationships analysed from a language use viewpoint

The purpose of this section is to analyse metamodelling from a language use perspective. The basic idea behind the analysis is, rather than to try to map domain languages and ontologies into the OMG four-layer hierarchy (see Figs. 2 and 4), we investigate first how language operates (Section 5.1) and how we can use that knowledge to propose an alternative metamodelling approach based on speech act theory (Section 5.2); and later show how the problems of the Orthogonal Classification Architecture that introduced the ontological/linguistic metamodelling paradox could be solved (Section 5.3). The analysis is performed using substantial and moment classes and by making a sharp distinction between the type and instance level of the class. At the type level, language constructs such as definitions, class names, attribute names and identifier names will be used. At the instance level, the terms attribute and identifier will be used – these give values to the names stated at the type level. It is important to distinguish between these different levels and the associated language constructs, because they are used to perform different language functions.

5.1. A language use analysis

To begin, the following speech act can be formalised into the simple table below (Table 1).

1. “Fido is a dog”

This speech act can be considered as an existential proposition that instantiates a substantial object using the identifier “Fido” in the Dog class. Consider that this is mutually agreed upon in this use context. We can then perform the following speech acts.

2. “Fido is 50 cm tall”
3. “Fido is brown”

As both speech acts refer to the previously instantiated object Fido, all three speech acts can be represented in Table 2, i.e. height and colour are attributes that ascribe moment universals to the substantial object. This is modelled in Fig. 10 in a UML diagram where the diagram is at the M1 Level (see Fig. 1). The ontological levels of O1 and O0 in Fig. 2 can be compared to the type level of the dog class (O1) with Fido being at the object level (O0).

However, the speech act below is more problematic.

4. “Fido is a Collie”

In this use context, Fido is already instantiated as a substantial object in the Dog class, so it would not make sense to instantiate

<table>
<thead>
<tr>
<th>Table 1</th>
<th>A tabular representation of speech act 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog</td>
<td>Dog's name</td>
</tr>
<tr>
<td>Fido</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2</th>
<th>A tabular representation of speech acts 1–3.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog's name</td>
<td>Dog's height (cm)</td>
</tr>
<tr>
<td>Fido</td>
<td>50</td>
</tr>
</tbody>
</table>
Fido as a substantial individual in a Collie class as well (as described in Table 3), because the purpose with this speech act is not to instantiate a new substantial object; it is to describe an already instantiated substantial object.

Considering our purpose, to describe the previously instantiated substantial object Fido as a Collie, Table 4 makes more sense (see Table 4).

If a UML class diagram were to be used to model this, then Collie would typically be modelled as a subclass of the Dog class. In other words, the subclass(es) of the Dog class (which include Collie, Labrador and Bulldog) are examples of the dog’s breed and could therefore, arguably, be represented as instances of a Breed class associated to the Dog class. However, this is clearly not altogether correct from a language use perspective because Collie is used in Table 4 solely as an attribute value in this case, and thus it is not used for sub-classifying a number of physical dogs. It is used to describe a substantial object in terms of its attributes. However, if the question “What is a collie?” is asked, the situation will be changed because one might have to answer the question by saying.

5. “Collie is a Dog Breed” (see Table 5)

In this context, it makes sense to instantiate Collie as a moment object of a Breed class and, if this existential proposition is agreed upon, we can then refer to and describe the Collie object in more detail with its own attributes (e.g. Table 6).

6. “A Collie is between 45 and 55 cm tall”
7. “A Collie is sable or tri-colored or brown”

This is modelled in Fig. 11 in a UML diagram where the diagram is at the M1 Level. Compared to the ontological levels of Fig. 2, the type level of the Breed class is still at the O1 level and Collie is at the object level O0. The reason for this is that Collie is a moment object, not a type. This can be compared with the old object-oriented design argument that an attribute and an association (to a second class) are virtually interchangeable. However, they are not, because, if Collie is used as an identifier, it is consequently used as a moment object of the breed class. In this case, the intent is not to talk about the attribute of an individual dog “Fido is a collie”; rather, the intent is to talk in general about what is common to all physical dogs of which the Breed name Collie is meaningful. The important conclusion is that the Breed class is used as a moment class, and its objects can now be used to sub-classify or partition the substantial objects of the Dog class. Thus, sub-classification can be modelled as a powertype pattern where the instances of the DogBreed Class create a set of parts from the instances of the Dog class (creating a mathematical partition)
learn is that we must make important decisions regarding which (appropriate) class concepts should be used, in order to communicate in a meaningful way in a certain context of use. We have to understand how different language functions and constructs are used in order to create a mutual understanding of the world, and we have to understand that the type–instance relationship is essentially a relationship between the intensional and extensional part of a class concept. As a consequence, a moment object like “Collie” (Table 7), cannot be a type in relationship to the substantial object “Fido” (Table 8), and a type, e.g. the intensional part (the definition) of the breed class, cannot be an instance of the intensional part (the definition) of the dog class. Furthermore, attributes are not used for sub-classification. In keeping with meaning theory, it is the class (sortal) construct that is used for classification. Sub-classification is accomplished by relating two classes to each other where the objects of the “powertype” class (e.g. Collie) are used to provide a partitioning rule, not a rule for how objects should be instantiated. The idea that there are metaclasses with instantiated classes that have objects of their own, as described in Fig. 2, creating a Breed–Collie–Fido instance-of chain of relationships, has no support in theories of meaning based on language use. One of the best-supported theories in the philosophy of language is namely that an individual can only be traced in connection with Sortal Universals, which provide a principle of individuation and identity to the objects to which they refer [23,48,64–68]. Moment objects like Collie are not sortals; thus, it is the superclass, in this case the Dog class (which is a sortal), that provides the rule for the instantiation of Dog objects.

5.2. A metamodelling approach based on language use

The analysis above implies that if we want to model the Breed–Collie–Fido relationship, we have to understand how these different language constructs are used in each context, how they relate to each other in that context, and which constructs are used to perform different language functions. In natural language use, these language functions are confounded, being used at the same time implicitly and explicitly. This has been seen (both recognised and ignored) in early modelling papers and books. We talk of a ‘Dog’ (a type) and of ‘Fido’ (an object) in the same breath. The type is similar to an intensional set definition whereas the individuals are in effect the members of that set, i.e., elements in the extension. Thus, we can divide the language use ‘level’ of Fig. 12 into three portions (but not separate levels in the multi-level architecture sense of Fig. 1): A, B and C. Level A, which includes the definition and constructs (the Keys) of the modelling language (which is the metamodel level M2 – Fig. 1), is of interest because it is important that the constructs used in the modelling language are explicitly defined. The language use levels B and C (which are both at model level M1) outline the language use of types, such as Dog, Breed (some of which are substantial types, some moment types), which are the intensional aspect of the extensions (the objects, which here may include Fido and Collie) underlying these types. This is the type–instance relationship. There is also a relationship between substantial classes and moment classes. At the type level, we can say that the breed is a powertype of the dog type. At the instance level, we can maintain that the moment objects are used for partitioning substantial objects and that the substantial objects have a functional relationship (functional dependency) to the objects of the moment class. There is a correspondence relationship between physical things (level M0) and the objects at Level C.

When humans communicate by talking and writing in their own native language, they use both the intensional aspects of language (Level B) and the extensional (Level C). However, most of the time, the type level is already understood. For example, if someone says, “Fido is a Collie”, they refer to an object that is an extension of

Table 7
A tabular representation of instances of the Breed class.

<table>
<thead>
<tr>
<th>Breed</th>
<th>Breed height (cm)</th>
<th>Breed colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collie</td>
<td>45–55</td>
<td>Sable, or tri-colored or brown</td>
</tr>
<tr>
<td>Labrador</td>
<td>50–60</td>
<td>Golden or black</td>
</tr>
<tr>
<td>Boxer</td>
<td>57–63</td>
<td>Brown</td>
</tr>
<tr>
<td>Golden Retriever</td>
<td>50–65</td>
<td>Golden brown</td>
</tr>
</tbody>
</table>

Table 8
A tabular representation of instances of the Dog class.

<table>
<thead>
<tr>
<th>Dog name</th>
<th>Dog height (cm)</th>
<th>Dog colour</th>
<th>Breed name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fido</td>
<td>50</td>
<td>Brown</td>
<td>Collie</td>
</tr>
<tr>
<td>Fran</td>
<td>58</td>
<td>Brindle</td>
<td>Boxer</td>
</tr>
<tr>
<td>Bran</td>
<td>55</td>
<td>Black</td>
<td>Labrador</td>
</tr>
<tr>
<td>Lassie</td>
<td>58</td>
<td>Tri-colored</td>
<td>Collie</td>
</tr>
</tbody>
</table>

Table 9
A tabular representation of instances of the Dog Collie class and the Collie Breed class.

<table>
<thead>
<tr>
<th>Dog Collie</th>
<th>Collie's name</th>
<th>Collie's height (cm)</th>
<th>Collie's colour</th>
<th>Collie's Breed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fido</td>
<td>50</td>
<td>Brown</td>
<td>Border Collie</td>
<td></td>
</tr>
<tr>
<td>Lassie</td>
<td>58</td>
<td>Tri-coloured</td>
<td>Rough Collie</td>
<td></td>
</tr>
<tr>
<td>Jackie</td>
<td>50</td>
<td>Tri-coloured</td>
<td>Border Collie</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collie Breed</th>
<th>Breed name</th>
<th>Breed height (cm)</th>
<th>Breed colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Border</td>
<td>48–56</td>
<td>Solid coloured, Bi-coloured, Tri-coloured, Brown</td>
<td></td>
</tr>
<tr>
<td>Rough</td>
<td>56–66</td>
<td>Sable, Mahogany sable, Shaded sable, Tri-coloured</td>
<td></td>
</tr>
</tbody>
</table>

wherein the instances of the Dog class refer to the instances of the DogBreed class. Tables 7 and 8 illustrate this.

Finally, we note that in a different context, where the universe of discourse is not about all dogs and breeds, say a collie-only kennel club context, it would be possible to use collie as a substantial class (Table 9) instead of a general dog class, and thus consider Fido as an instance of a (Dog) Collie class. The defining predicate of that class would then be “A domesticated carnivorous mammal (Canis familiaris), which are all of the subspecies Collie,” i.e., this substantial class would have another meaning than the general dog class in Table 7. However, the objects of this (Dog) Collie class are still used to identify and represent individual attributes of instances of physical dogs like Fido, not to talk about several instances of physical dogs.

Furthermore, in such a universe of discourse there would be no other breeds of dogs of interest except Collie, and thus it would be meaningless to use the general Breed concept as a moment class (Table 7). However, it would be of interest to partition the substantial Dog Collie class into, say, Rough Collie, Border Collie and Smooth Collie, thus creating a moment class Collie Breed with the defining predicate “Groups of closely related and visibly similar collie dogs” that makes that partitioning meaningful (see Table 9).

Essentially, the point we want to make is what counts as a valid (and meaningful) object depends on the constitutive rules agreed upon regarding the particular institutional context in which it is being instantiated and used. The important thing that we have to
the dog type without mentioning the class names or attribute names "dog" and "breed". The native speaker and listener already understand this implicitly. This is obvious because Collie does not mean any object (e.g., a big cat) that is brown or tricolored and between 45 and 55 cm tall. In this use context, it must mean a dog that fits this description. It is only when communication breaks down and we do not understand one another that we have to start using the class and attribute names at the type level to explain what we mean (as in speech act 5) and, if this is not enough, we even have to try to define the constructs at the type level. When people talk or write about ordinary objects in their daily life, they do not have to consult a dictionary, thesaurus or a grammar in order to understand what "dog" or "breed" means and how these constructs should be used. However, if we want to analyse in a formal way how people communicate, we have to be more explicit and precise about the classes used, their relationships, and the type and instance level. This is shown, for example, in the tables above where the speech acts performed have been analysed in tables explicitly describing the dog and breed classes and their relationships. The type level of each class has also been analysed using the identifier and attribute values in the rows of the tables. Being explicit about the language constructs used in a use context is arguably at the core of conceptual modelling. The reason is twofold.

First, IT systems are always developed with a purpose in a particular use context. Thus, we have to be explicit about the classes that fulfil our purpose. In such a context, the cursory use and understanding of certain important classes is inadequate. For example, if we want to develop an IT system for a Kennel club, we need a Dog class to instantiate and describe individual dogs, and we need a Breed class to sub-classify (partition) the dog class. In such a context, an explicit model and a definition of the dog and breed classes is necessary in order to determine which physical dogs should be allowed as registered dogs of the Kennel club. It may, for example, be important that the dog is a purebred. The definition of breed will likely cause debate and an important part of the modelling process will then be to achieve consensus on this matter. The result of such a negotiation process cannot be seen as a shared mental state, or set of interdependent ideas that carry the meaning of the dog and breed concepts as an abstract system. It should rather be seen as the articulation of a number of linguistic and social rules that regulate what are valid instances (objects) of the dog and breed classes. These rules and objects constitute part of the social world as a system of institutional and brute facts. Such a system is not to be found in the physical world or in a mapping of it onto mental states. Institutional objects, or facts, are objects with deontic powers (rights and obligations) that are valid in a particular social context [43,69,70].

Second, IT systems are formal systems that operate according to explicit rules. The classes needed for enabling communication using the system have to be explicitly described and declared. These rules are implemented in the system and are used to instantiate corresponding objects, and should thus be formally defined. Typically, dog and breed tables described (Tables 7 and 8) have to be formalised into databases and programs in order to develop the Kennel club IT system. We need a data type level definition of the database table that should be used to store information about the Kennel club's dogs. How far one goes in formalising the rules of the language depends on the design and use situation. Although a Dog class is needed in the model, a fully formalised definition of the principle of application (see Section 5.1) may not be needed, assuming that the user(s) know what a dog is. However, a more precise partition rule must be agreed upon if, for example, only purebreds are allowed. This may not be so easy to decide and will require that the dog breed class must be defined with precision at the type level as well as the instance level. Such a political process, rather than a formal logical process, may also involve deciding which standard to use that defines the notion of Dog Breed.

Importantly, modelling languages must be able to bridge the gap between informal and formal languages in order to enable implementing the rules of a domain language in an IT system that could be used to perform speech acts (communication acts) in a certain context of use [44]. Thus implicit assumptions have to be made explicit, and be brought to the forefront in order to be defined and agreed upon. This also implies that the metamodels used to define and analyse modelling languages should also be based on an understanding of natural language use, and not only on formal languages.
5.3. Solution to the Orthogonal Classification Architecture (OCA) problems

The purpose of this section is to show how the problems with the Orthogonal Classification Architecture (OCA) (left side of Fig. 13 – returning us to our first example domain of horses) can be resolved from a language use viewpoint i.e. levels B and C (right-hand side of Fig. 13).

The first problem is that the OCA model does not include an explicit horse class or horse concept. Thus the meaning of “Prancer” as an instance of a Horse is undefined; see the discussion about identification in Section 4.3 (Fido is a dog) and Section 5.2 above, where it is made clear that if Prancer is to mean an instance of a horse, a class is needed. The omission of Horse also means that Breed could be any Breed because no restriction is set on the Breed class by the Horse class, which consequently means that Breed means any Breed. Clearly, the model relies heavily on implicit knowledge and assumptions. However, we can understand why the Horse class is left out, because this could mean that there would be four ontological levels needing explanation, perhaps with the horse concept being a linguistic instantiation of a metamodel class. However, leaving out the Horse class in the model is one reason for the linguistic/ontological paradox. It is symptomatic of a confounding between Horse and Breed, seen more obviously in some software engineering papers where, for example, Task is confused with TaskKind.

The second problem is that it has an invalid double instanceOf: one from Prancer to Shetland Pony and one from Shetland Pony to Breed because a concept cannot be an instance of another concept and a class cannot be an instance of another class. The language use analyses in Section 5.1 showed that Collie (or ShetlandPony) could be transformed from an attribute (adjective) into an object (into a proper noun). This is called nominalisation in linguistics, which means that ShetlandPony could be used as an attribute in one class (Horse), and as an object in (an instance of) another class (Breed) but with different meanings. Colloquial use of the same word (e.g. ShetlandPony and Collie) in two different meanings accentuates the problem. Using the collie example, we could think of the (apparent) subtype of Dog as CollieDog to more explicitly distinguish it from Collie as a Breed. In software engineering this is recognised by the use of Task and TaskKind, etc.

Furthermore, the object ShetlandPony cannot be a class in relationship to the Breed Class, because the language function of ShetlandPony in the Breed Class is to instantiate and refer to a unique object, not to classify it—this is the function of the class name (Breed). A major problem in the metamodelling literature is that the inter-layer relationship of instanceOf is often regarded as sacrosanct in object modelling [e.g. 51]. However, the instanceOf relationship is not transitive—it is a type–object relationship within the intensional and extensional parts of a class. There is, for example, a partition hierarchical relationship between breed objects and horse objects, but there are no hierarchical instance-Of relationships between different ontological levels as described on the left hand side of Fig. 13. If we accept ShetlandPony as an object i.e. a moment individual of the Breed class, we cannot accept Prancer as an instance of ShetlandPony. This was explored in [14], in which the authors propose that developers cognitively invent a class–object pair when dealing with an M1 class that they need to be both conformant to a metamodel class and also able to be used to create M0 level objects (see earlier discussion of Fig. 3).

The third problem is that an object in the OCA model is only considered to be a representation of a physical thing; thus, there is a 1:1 relationship between the physical thing and the object. An object is a language construct. Before it can be referred to it must be instantiated so as to correspond to real world things, such a correspondence does not imply a 1:1 relationship, it may imply a 1:M relationship as the correspondence relationship between ShetlandPony and the physical things in Fig. 13.

![Fig. 13. The traditional ‘ontological metamodelling’ approach (left hand side – cf. also Fig. 2) and a more correct model that acknowledges the substantial universal and moment universal nature of the entities involved.](image-url)
Considering these problems from a language use viewpoint, we can redraw the left-hand side of Fig. 13 as done in the model on the right-hand side. The language use layer is at the M1 layer (Fig. 1), and although we have indicated the ontological levels (O0 and O1) within the M1 layer there is no need for them. This contradicts the claim made in [71], where the authors claim that two-level modelling, only allowing the O0 and O1 levels, forces modellers to use artificial workaround mechanisms in order to model problems that need multiple ontological layers i.e. the O2 and beyond. The right-hand side of Fig. 12 shows that this is not the case. This model includes a substantial horse class where Prancer is a substantial object of that class. A clear separation is made between objects (at the M1 level) and real-world things. Objects are language constructs; physical things correspond to brute facts. The introduction of the Horse class implies that there are two type–instance relationships, which solves the issues with the double instance of problem of the left-hand side of Fig. 13. However, there is also a partition relationship between the Horse and Breed classes at the type level, and there is a functional relationship from each substantial horse object to one moment object in the Breed class. There are also correspondence relationships between one substantial horse object (Prancer) and one physical thing, and between one moment object (Shetland Pony) and a number of physical things.

### 6. Other proposed solutions

A number of scholars have previously attempted to analyse and explain the linguistic/ontological paradox that arises from OMG’s four-layer hierarchy (Fig. 1) and the Orthogonal Classification Architecture (Fig. 2). In this section we will reassess these solutions especially with respect to the views propounded in [33]. These authors deduce correctly that the distinction between the intension and the extension of a class is important to consider. Hence, Gašević et al. [33, p. 11] conclude that “a class can be considered an abstract system composed of intensional and extensional [sic] parts which are also abstract systems”.

However, turning our attention to the ‘solution’ to the linguistic/ontological paradox presented in [33] and shown in Fig. 14, we can see that:

- At the O1 level is an abstract system, Collie:AS, which represents (μ) the meaning of the real world concept, which must reside in a collective mind (it is hard to tell what the light bulb means, but we assume that it is a collective – rather than an instance – idea). The intensional part represents the definitional phrase: “has long hair, has bushy tail, can herd sheep” in the real world. Collie is at the same time an ElementOf (ε) the set of Breed, and Collie is ConformantTo (χ) the intensional part of Breed:AS, at the O2 level.

- At the O2 level there is an abstract system, Breed:AS, which represents (µ) no real world concept, and the intensional part has no definition in the real world.

The model above can be criticised as being somewhat unclear and incomplete. For example, there is no definition of the breed class; it is unclear what is meant by the light bulb; and it is unclear what the abstract system Fido:AS, means. The latter is not a class because in order for it to be a class it has to have both an extensional and an intensional part (see the definition above). However, Breed:AS and Collie:AS must both be classes, and we will concentrate on three major problems with their solution.

At first, Gašević et al. [33] claim that “in ontological metamodelling the instantiation relationship (instanceOf) happens between two concepts when one concept is ConformantTo (χ) the intension and is an ElementOf (ε) the extension of the other concept.” Thus in Fig. 14, there is an instanceOf relationship between the classes of Breed and Collie. As we noted earlier, the relationship between two concepts/classes can never be a type–instance relationship. The authors do not recognise that the relationship between concepts at the type level is a powertype relationship, not an «InstanceOf» relationship. It is also clear from the figure that Fido is an instanceOf the Collie class. Our observation is that Fido, which is a substantial object, cannot be an instance of Collie (which is a moment object) but rather must be an instance of a Dog class. The lack of a Dog class in Fig. 14 is a major problem (as in most models that try to solve the linguistic/ontological paradox). In the model, it is assumed that the dog named Fido has a meaning without there being a dog type – which is impossible (see Sections 4.3 and 5.1). The Dog class is implicitly assumed but not explicitly modelled, which leads the reader’s interpretation astray. If the Dog class had been modelled, as it should have been, it would have to reside at the O3 level. The Breed:AS should then be modelled as an ElementOf (ε) the set of Dogs, and at the same time ConformantTo (χ) the intensional part of Dog:AS. The problem is that there are no hierarchical ontological levels and relationships as described in Fig. 14.

![Fig. 14](https://example.com/fig14.png)

Fig. 14. The relations between the instantiated elements and the intensional and extensional properties of their super classes - as proposed in [33]. With kind permission from Springer Science and Business Media.
Secondly, the authors claim that “two classes may have the same class extension, but still be different classes.” This can be demonstrated as being incorrect since a type–instance relationship must always be between the class type level and instance level of the same class, because it is the intensional part of the class (the defining predicate, the principle of application) that defines which objects can be instantiated in the class. A physical thing can of course correspond to two different objects in different classes in different ways. For example, a physical substantial thing (dog) can (1) have a one correspondence relationship to the Fido object of the substantial Dog class, and (2) have a another correspondence relationship to the Collie object of the Breed class. But the substantial object Fido cannot belong both to the substantial Dog class and moment Breed class in the same domain language because these classes have different defining predicates. It would not be meaningful to state that “Fido is a Breed” and “Fido is a Dog”.

Finally, the issue of how Breeds other than Collie should be modelled is neglected (which is the practical side of the problem described in Section 3), thus ignoring the explosion of classes at the O1 level that will inevitably follow from the solution presented in Fig. 14, which makes it difficult to use in a practical sense.

Another option to solve the linguistic/ontological paradox is to create a chain of type models [5]. A type model of a type model is, however, a powertype [35,72]—a construct eschewed by Atkinson and colleagues [20,51]. In general terms, the relationship labelled type–instance (in Fig. 14) must mean that the Breed is a powertype [14] and depicted as a software pattern in [73] see Fig. 15.

The set-based analyses in Section 3 (see e.g. Figs. 6 and 7) show that the OCA hierarchy requires the notions of clabject and powertype. For example, if Collie (or ShetlandPony) is an ontological instance of Breed, i.e. the object facet of the clabject construct, then this could only mean that Breed is a powertype, so that each element of Breed maps to a part of the Dog set (or Horse set). So what seems to be a nice hierarchical model hides the dependency of the powertype pattern (see Fig. 15). Although the powertype pattern is an improvement because it recognises the need of the Horse class, it is inadequate in solving the problem, because it does not make an adequate distinction between the type and instance level of the class. In these models, the instances of the breed class (e.g. Collie or Shetland Pony) are described both as a subtype and instance at the same time (a clabject), which is impossible because the type and instance distinction of a class cannot be merged according to language use theory. The clabject construct is impossible from a language use perspective because if Collie is an instance in relationship to Breed it cannot be a subclass in relationship to Collie because if Collie is used within the Dog class it must be used as an attribute (see Section 5.1). A consequence of models that do not distinguish between the use of terms like “Collie” as attributes, or objects, is that it seems as if “Fido” is instantiated by the Collie subclass not the superclass “Dog”, which is similar to the problem of Fig. 14. However, instead of an explosion of classes you get an explosion in subclasses (see Section 3 and Fig. 7).

7. Metamodelling based on language use

The analyses of the metamodelling literature presented in this paper have identified a number of problems. A basic problem is that the architectures (OMG and OCA) analysed do not provide any explicit theoretical foundation based on foundational ontologies or meaning/language theories. In particular, they do not relate to language use theories. It is also unclear how these architectures relate to set theory and formal languages used in mathematics, programming and database languages. This is the cause of a number of other key problems in linguistic and ontological metamodelling:

- No explanation exists of how entities in the architectures relate to the real world, which is also recognised as a key problem in [18].
- The architectures implies that all relationships between elements are type–instance relationships [18].
- No explicit principle exists for judging at which level a particular element should reside.
- Proposed architectures, solutions and design patterns do not correctly distinguish between type level and instance level of classes.
- The pragmatic meaning of sub-classification is not well understood.

These problems lead to the need for a new (meta)modelling paradigm that is at least relevant to conceptual modelling in the context of software engineering and information systems development. We suggest that, inter alia:

- In bullet form for clarity

---

![Fig. 15. A Powertype pattern showing relationships between Breed, Dog, Collie and Fido.](image-url)
7.1. A solution to the linguistic/ontological paradox

Rather than offering a simplistic statement that ontological (meta)modelling creates token models, whereas linguistic modelling utilises type–instance semantics, we seek a more theoretically sound basis. Based on the assumptions described above, we propose a solution for the linguistic/ontological paradox as exemplified in Fig. 16.

In the model depicted in Fig. 16, Fido and Lassie are instances of Dog with two corresponding real (physical) dogs and an associated Breed object Collie – all of which are at the instance level (in Fig. 15 called O0). Both the Dog objects (Fido and Lassie) and the Breed object (Collie) are instances of their respective classes in the type level (previously called O1). Here, Breed is a moment class related to the Dog class, neither of which requires an additional (O2) level for their definition other than the fact that both are linguistically defined as classes via an M2 Class class. The idea of needing an ontological metaclass is obviated. From a language-use perspective there are no hierarchical ontological levels such as those described in the OCA architecture (see Fig. 2). In other words, so-called ontological metamodelling has been shown to be simply regular ontological (or domain-specific) modelling, i.e. the model is akin to a domain ontology (see also [53]). Thus, the misdirection in diagrams such as Figs. 2 and 14 is a combination of the omission of the Dog class (a base class) and the fact that Collie and Fido are both objects, and that the mapping from Dog objects to Breed objects is a functional relationship and not instantiation – as seen in Fig. 16. The advantages provided in this new, language-use-based approach to modelling are listed in Table 10 as a point-by-point rebuttal of the claims of [33].

However, it should be noted that, while the right hand side of Fig. 15, which depicts a powertype solution, shares some similarities with Fig. 16, powertype-based models do not make an adequate distinction between the type and instance level of the class. The
A further criticism from a language use perspective of the powertype solution of Fig. 15 is that it cannot depict the fact that the
Breed class has attributes of its own that are distinct from the attributes of the Dog class. Furthermore, the model does not show that each object of the Breed class partitions the objects of the Dog class i.e. creates different parts within the partition. In Fig. 15, it also seems as if there is an inheritance relationship between the Dog as a superclass and all the subclasses, which should all have different attributes at the type level. However, they do not – because they cannot be subtypes (they are equivalence classes or subsets [74, p. 80] [39, p. 25] i.e. parts of the partition) because Collie can only be an attribute in the Dog class (the act of predication) and attributes cannot be used to create subtypes. As noted earlier, it would furthermore be very difficult to model a powertype with a large number of subclasses using such a notation, making this approach potentially impractical.

Additionally, the consequences of the language-use approach are context dependent, as noted earlier and when it comes to so-called “ontological metamodelling”, there is no need to introduce metaclasses (metaconcept), and there is no universal rule that can tell us what is appropriate.

The important thing that we want stress is that modellers need to understand how different language functions and constructs are used in order to make sense of the world, and what contextual assumptions are made in different domain modelling situations. There is no such thing as a universal domain model that needs assumptions are made in different domain modelling situations. There is no such thing as a universal domain model that needs to understand how different language functions and constructs are used in order to make sense of the world, and what contextual assumptions are being made, i.e. how the model relates to the world. Without this, one gets trapped in ambiguity and contradictions.

7.2. A tentative metamodelling architecture based on language-use

Excluding the idea of ontological metamodelling levels, we can now reformulate the architecture based on Fig. 1. Rather than starting with Fig. 1 and asking how can one link modelling languages to existing metamodels [53], we start with Fig. 16 and ask how we can develop modelling languages that are based on the language use perspective. The answer is straightforward – as shown in Figs. 12 and 16. The ‘language definition’ must include constructs such as class, type, attribute, identifier, partition, and physical thing, in order to represent different language functions and to be able to distinguish between things and objects (Fig. 17). The M2 level must also include constructs that can be used for representing the relationships between these constructs. This can be exemplified by the M2 level of Fig. 16. These can be seen as a brief model of the modelling language used and could be further developed into a more comprehensive metamodel. The metamodell would then be linked to a foundational ontology and a theory of meaning (see Section 4). Thus, the definitional M2 layer must include an appropriate foundational ontology [42]. The models produced at the M1 layer with the modelling language must conform to the definition of the modelling language at layer M2. The models produced at the M1 layer should also correspond to physical world entities (things), which constitute the M0 layer. The relationship between the objects in the model and brute facts is a correspondence relationship (Fig. 12), which means that models at the M1 level are used for representing and describing the things, properties and events of the material world. The relationship between models and institutional facts is that a model can refer to already instantiated institutional objects but also that there can also be conformance relationships e.g. to standards, regulations and statutes.

7.3. Practical relevance

The problem of not having a good way of explaining the “ontological/linguistic metamodelling paradox” is a problem of great
The aim of the OMG four-layer hierarchy and the Ontological Classification Architecture (OCA) is to support the development of modelling languages, domain modelling and requirements engineering. Thus, it is problematic that these basic architectures cannot be used to accurately explain the relationship between domain models and the world, how modelling languages could be related to ontologies, and how ontologies could be used to support domain modelling.

It is not without reason that multi-level ontologies are ambiguous and contradictory, as we have shown in Section 7.1. This confusion could also be exemplified by this cry for help on the Internet [75]:

"I'm trying to comprehend ontology basics. Here's an example:

  car (class)
  2009 VW CC (sub-class or instance?)

My neighbor's 2009 VW CC (instance)

My issue is understanding what is "2009 VW CC" (as a car model). If you're making product model a sub-class in the ontology - all of a sudden your ontology becomes bloated with thousands of subclasses of a "car". That's redundant. At the same time we can't say "2009 VW CC" is an instance, at least it's not material instance of a class. Does it make sense to distinguish between regular instances and material (distinct physical objects)?

At the other hand, if both are instances (of different nature so to say), then how can instance inherit properties / relations of a non-class?"

Apparently, there is confusion among modellers and programmers about how product names (e.g. "2009 VW CC") are typically used. Are they types, instances or both types and instances (classes), and how should they be modelled in relation to cars?

This is an example of a Product Lifecycle Management problem, common in information systems. Cars and car models are one typical example [76], a second (that will be used here) being that of computer hardware [71]. Both are elaborations of a practical situation of the 'Item Description' pattern [77], which is a pattern commonly found when modelling the requirements of enterprise information systems, product lifecycle-management systems, and product databases.

We do not claim that this is always the most significant problem in any given modelling situation, but it is one that we can solve, as discussed in this paper.
adds unnecessary (i.e., accidental) complexity to domain models because it restricts the domain model to the use of two levels only, e.g. the M1 and M0 levels of the OMG four-level hierarchy (see Fig. 1). To resolve to this problem, the authors present a solution based on the OCA and the idea of clabjects [78].

The authors claim that the OCA (Fig. 2) could be used as an underpinning modelling infrastructure, since it explicitly acknowledges the existence of multiple ontological domain levels. However, in order to accomplish this, the notion of clabject is needed. Using clabjects as their key to solve the “Item Description” problem, they [71] claim that multi-level ontology modelling could be used as an underpinning modelling infrastructure, since it explicitly acknowledges the existence of multiple ontological domain levels, and supports them directly with a uniform notation for all the levels and with a minimum of modelling concepts. The solution to the problem is presented in Fig. 19, based on Fig. 12 of [71], although their model has been modified to explicitly show the clabjects and the ontological levels.

In order to make the model work, another special feature called “deep instantiation” [71] is used together with clabjects. In this approach, attributes have an additional property called potency. The value of potency is decremented whenever it is instantiated. When potency equals zero, no further instantiation is possible. For example, the subtype “ComputerModel” has a field “price” of potency two, indicated by the superscript “2” as part of the field name. When “ComputerModel” is instantiated to create “ComputerStandard” (actually a clabject), all of its fields with a potency higher than zero are copied to “ComputerStandard”, with their potencies reduced by one. Thus “price” becomes a field of potency one, while “processor” becomes a field of potency zero (i.e., a slot). The further instantiation of “ComputerStandard” to generate the individual “c1” reduces the potency of “price” to zero and turns it into a regular slot. Thus the processor capacity could be represented (i.e. have a value) at the clabject level but the price only at the physical object level. The authors claim that this version of a domain model that solves the “Item Description” pattern is not only the model with the “least accidental complexity it also is the most expressive.” The authors exemplify this by stating that a number of variations of the “Item Description” pattern have been proposed using e.g. powertypes and stereotypes, which only reflect workarounds that obscure the meaning of the domain model. However, despite protestations in [71] that powertypes provide a more cumbersome resolution, as we showed in Fig. 14, the smooth multi-level hierarchy in Fig. 19 is in fact an implicit powertype pattern. In the OCA solution (left hand side of Fig. 20) we see that ComputerStandard...
(a clabject) is an ontological instance of ComputerModel (a class). This can only be valid if ComputerModel is a powertype, such that each element of the ComputerModel set maps to a part of a Computer set when there is a partition. The ComputerModel set in this example has only two elements (ComputerStandard and ComputerDeluxe) and each ComputerModel element supplies a name to each of the subclasses. Note also that this OCA solution (left hand side of Fig. 20) does not include a class to represent a Computer per se. As we have noted above, omission of the base class in a domain is problematical and often the main source of the perceived ambiguity. Another ambiguity with the model is that two names have to be used to represent each individual clabject. For example, “ComputerStandard” is used to represent the type facet of the clabject and “PC Standard” represents the object facet.

7.3.2. The powertype approach

As noted in [71, 76], powertypes have also been utilised for this kind of problem. Continuing with the computer hardware example, in Fig. 21 we see the result of applying the powertype ideas of Section 3.

However, there are also problems with the powertype model. The first problem is that there is no agreed notation that indicates that the ComputerModel class is a powertype used to partition the Computer class, because it is marked only as an ordinary association relationship. To quote Halpin [79], “We may not assume that a binary association from an object type to a type marked in some way as a powertype is of this nature.” Another problem is that it seems as if the subclasses or clabjects have different attributes (at the type level), which is itself misleading because the ComputerStandard and ComputerDeluxe have only different attributes at the instance level. It would also be very difficult to model a situation where there are a large number of subclasses using such a representation, since in the real world there is a plethora of ComputerModels. Furthermore, the model does not explicitly show that each individual of the ComputerModel class partitions the individuals of the Computer class into different subsets, and it seems (incorrectly) as if an instance of a Computer (C1) is an instance of a ComputerStandard and, in order to fix this problem, the clabject construct has to be used. The clabject becomes a construct that is placed inbetween the M1/M0 level, thus blurring the important distinction between the type and instance level.

7.3.3. The language use approach

In this section, the “Item Description” pattern [77] is examined further in order to argue for the practical relevance of the ontological/linguistic metamodelling paradox outlined in Section 2. A solution to this important practical problem is presented in Section 7.3.1 using the Orthogonal Classification Architecture (OCA) (see Fig. 19) and, in Section 7.3.2, using a powertype pattern (Fig. 21). The analyses showed that neither the OCA solution, which is based on an implicit powertype pattern, nor the powertype pattern itself could solve the problem properly. We will now show how this problem can be solved satisfactorily using the language use metamodelling approach presented in Sections 7.1 and 7.2 (see Fig. 22).

Fig. 22 shows that ComputerItem and ComputerModel are two distinct classes. Ontologically speaking, ComputerItem is a substantial or base class and ComputerModel is a moment class. As a consequence, we note that:

1. The notion of clabject, which resides between the M1/M0 levels, is not needed. Instead, a clear distinction has been made between the type and instance levels within the M1 layer.
2. Constructs like deep instantiation and potency are not needed.
3. The model shows clearly that the ComputerItem objects “C1” and “C2” are instances of the ComputerItem class and not instances of clabjects “ComputerStandard” and “ComputerDeluxe”.
4. The model shows that the ComputerModel class and the ComputerItem class restrict each other through a partition relationship.
5. The attribute computeritemmodelname is used to describe an instance of a ComputerItem, and the Computermodelname identifier to instantiate and refer to an instance of a ComputerModel.
6. There is a functional relationship between ComputerItem objects and ComputerModel objects.
The objects of the ComputerItem class are clearly separated from the things they represent and there is a correspondence relationship between these constructs.

In terms of sets, we can maintain that the ComputerItem class in Fig. 22 can be represented as a (partitioned) set with two parts (Fig. 23), but that the relationship between the two sets is not a functional relationship.
classification relationship (see Fig. 6). Thus the elements of the powertype are not objects with their own instances, which would mean that they are clabjects, which is an impossible construct from a language use perspective. As soon as we realise this, and we understand that things and objects are not the same, which implies that objects do not have to have a 1:1 correspondence to physical things, there is no need for clabjects, which actually are moment objects. Martin and Odell [38] have shown that the powertype pattern is a design pattern with high practical relevance. However, this design pattern cannot be understood correctly without an understanding of language use and a foundational ontology.

The solution described above solves the Item description problem in the form of a general design pattern. However, we do not claim that the model fits each and every design context, because there are no universal domain models. For example, the model above assumes that price is set at the model level not the item level, although the model could easily be adapted to represent such a design situation by adding a computeritemprice attribute to the ComputerItem class. Furthermore, product models could be extremely complicated and could contain numerous levels but, as soon as the basic design principle is understood, designers could be helped to model better and to understand a multi-level product structure using the metamodelling approach presented in Sections 7.1 and 7.2. In order to help modellers to understand that objects are language constructs and that physical things are material entities, it needs to be observed that there are different types of objects: substantial and moment objects, and that they correspond in different ways to real-world things. A design pattern like the one presented in Fig. 23 could be very helpful. To solve these kinds of problems, the focus of metamodelling is typically what the designers of abstract ontologies intentionally avoid; to quote [71, p. 345]: "Our "universe of discourse" is therefore not a set of physical entities from the real world but contains elements that were created in a conceptualization e.g., by domain analysts. Measuring the adequacy of domain models with respect to a conceptualization, as opposed to the real world, is useful in order to avoid philosophical arguments of whether universals … should be assumed to exist in the real world and what constitutes an appropriate model of an excerpt of the real world."

Based on the design pattern presented in Fig. 23, we can therefore give advice to modellers and advise programmers not to create subclasses for each moment object (e.g. “PC standard”, “2009 VW CC”), because, in most cases, the operations on the moment objects are identical and the instances of both the substantial and moment classes should be recorded in the database, not as subclasses in the code.

7.4. Complexity and learning

Metamodelling is indeed a complex task that requires somewhat intricate tools, although we also recall and superpose the so-called Einstein’s Razor [80, p. 42]: “Everything should be made as simple as possible, but not simpler”. A consequence of our proposed solution is that modellers would have to be trained to learn some basics about foundational ontologies e.g. the distinction between concrete (physical) things and conceptual objects [47]. Things have intrinsic (physical, substantial) properties, conceptual objects do not; they are characterised by attributes defined by humans. Insofar as an object is an instantiation of a class using a specific language, objects are thus conceptual language constructs with a semantic and pragmatic meaning [23]. Furthermore, the identity of an object has to be socially constructed using an identifier or a definite description [43]. As a consequence, one object could only be instantiated in one class, but one thing could correspond to many objects. Modellers have to learn that the physical world has no inherent meaning unless people, using a language in communication, assign meaning to it. Thus, meanings have to be socially constructed and have to be understood in a social context, whereas sentences are used in communication acts, using a number of general language functions (classification, instantiation, reference, description). Modellers must also understand the distinction between predication (description) on the one hand, and inheritance and sub-classification on the other. Attributes are used to describe individual objects, not to sub-classify them. This is a severe misunderstanding in the modelling techniques commonly used. Such an ontological and epistemological foundation would simplify modelling by avoiding fixes such as clabjects, potency, deep characterisation.

Fig. 24. Solution to the “Item Description” pattern without the M0 level.
Once modellers have learned (1) the distinction between the M1 model layer and how it relates to the real world at the M0-level, (2) and the distinction between the type and instance level of a class within the M1 layer, there is no need to explicitly retain the M0-level in domain models or the container box that keeps the type and instance levels of a class together within the M1 layer, which makes the domain models simpler (see Fig. 24).

Thus, we can use design patterns as shown in Fig. 23, more as a pedagogical tool for learning how different objects at the M1 level relate to the world, and only use the M1 level when the domain language is modelled. The modelling technique is also scalable, which is shown in Fig. 25 where an additional power-type “Brand” has been included, partitioning the ComputerModel class.

Once the modellers have learned these modelling principles, the models could also be simplified only showing the type level (Fig. 26).

Another advantage with this proposed framework is that we could stick with traditional two-level modelling within the M1 layer. Thus software modelling tools could be developed, or existing tools could be adjusted, in order to support the necessary modelling functionality without additional workarounds and fixes.

8. Conclusions and future work

In this paper, we have analysed typical modelling examples [5,18,33] by using speech act theory and foundational ontological concepts to show how semantics (linguistic meaning) can be explained in terms of pragmatics; i.e. how sentences are used in acts of speech. Based on this analysis, we have resolved the linguistic/ontological paradox, which is the main contribution of the paper. We have also proposed a tentative metamodelling architecture based on language-use, thus offering a contribution to the foundations of conceptual modelling.
We have proposed a solution of the linguistic/ontological paradox by identifying (i) the lack of substantial classes (such as Dog and Horse). Based on foundational theories, language use and set theory, we have shown the error of omitting these classes, and (ii) the misuse of a moment objects (e.g. Collie and Shetland Pony) as if they were substantial classes or clabjects. Furthermore, (iii) we have identified that the substantial classes often have powertype relationships to moment classes, making this a general design pattern. The powertype design pattern (iv) has been remodelled based on the insight that moment objects are used to partition substantial objects. Thus, the moment objects are not used for subclassifying substantial individuals but as names for parts in the partition of substantial objects. The model (v) in Fig. 16 (vi) avoids the need for the clabject construct, which is an infeasible construct in a theory of meaning based on language use. Fig. 16 (vii) also avoids the explosion of subclasses, which is a feature of traditional powertype models. The reason for this is that there is no attribute inheritance from the substantial class to the objects of the moment class, as described in traditional models. Altogether, the idea of sub-classification based on attribute inheritance is impossible from a language use meaning point-of-view, because attributes are constructs used for describing individual objects not for sub-classifying them, as we noted earlier.

In the paper, we have identified that the OMG four-level and OCA architectures, which should support the development of modelling languages and domain modelling, are not based on a foundational ontology and are at odds with language use theory. This is the cause of a number of key problems in linguistic and ontological metamodelling:

1. No explanation exists of how entities in the architectures relate to the real world, which is also recognised as a key problem in [18].
2. The architectures imply that all relationships between elements are type-instance relationships [18].
3. No explicit principle exists for judging at which level a particular element should reside.

In this paper, we have presented a tentative metamodelling architecture: (1) that explains how the M1 level relates to the world at the M0 level; (2) that the type-instance relationship is only a relationship between the type level and instance level of a class at the M1 level; and (3) that gives an explanation regarding at which level a particular element should reside: at the M2 definitional level, we should find a foundational ontology and definition of modelling constructs; at the M1 level we find the domain models, and at the M0 level the real world.

In conclusion, it is vital for quality modelling that the ideas presented above are fully integrated into set theory representations of models and metamodel architectures, although we note that the architecture used in Fig. 17 is far from complete. For our future work we propose to:

- Develop a more elaborate ontology and modelling constructs for the new modelling approach.
- Investigate in depth the use of mathematics and set theory to formalise the model into a computable representation.
- Investigate how we can build more efficient and meaningful systems by combining language use theories of meaning and mathematics.

The ideas presented in this paper may at first sight appear to be complex. However, they are in fact far less complex than the suggested alternatives, such as the OMG multi-level and OCA architectures. As John Maynard Keynes is known to have said, “The difficulty lies, not in new ideas, but in escaping from the old ones, which ramify, for those brought up as most of have been, into the corners of our minds.” Essentially what we suggest is a change in mindset similar to the “pragmatic turn” in philosophy [81,82]. Our focus as modellers needs to shift from models to language, and by extension to how language is used to achieve pragmatic outcomes [46].

Acknowledgements

We wish to thank the extensive and very helpful comments of the anonymous reviewers of the original draft of this paper.

References


