Demystifying Ontological Classification in Language Engineering

Colin Atkinson[1] and Thomas Kühne[2]

[1] University of Mannheim,
B6, C2.11, Mannheim, Germany
atkinson@informatik.uni-mannheim.de
[2] Victoria University of Wellington,
P. O. Box 600, Wellington 6140, New Zealand
Thomas.Kuehne@ecs.victoria.ac.nz

Abstract. The introduction of ontological classification to support domain-metamodeling has been pivotal in the emergence of multi-level modeling as a dynamic research area. However, existing expositions of ontological classification have only used a limited context to distinguish it from the historically more commonly used linguistic classification. In important areas such as domain-specific languages and classic language engineering the distinction can appear to become blurred and the role of ontological classification is obscured, if not fundamentally challenged. In this paper we therefore examine critical points of confusion regarding the distinction and provide an expanded explanation of the differences. We maintain that optimally utilizing ontological classification, even for tasks that traditionally have only been viewed as language engineering, is critical for mastering the challenges in complex systems modeling including the validation of multi-language models.

Keywords: ontological classification, linguistic classification, semantic classification, language engineering, metamodeling, multi-paradigm modeling

1 Introduction

Ontological classification was originally suggested in conjunction with linguistic classification as part of a dual classification scheme to address inconsistencies in the original four-layer architecture associated with the UML [8]. Without an improved understanding of the precise nature of the four layers and the relationships between them, it was not possible to reconcile the intended linear organization of the layers [28] with the overall architecture's claims to strictness [3]. Recognizing two different classification principles and combining them in a dual classification architecture turned out to be the key to allow strictness to be enforceable in two orthogonal dimensions [7].

Being explicit about the ontological and linguistic classification dichotomy also proved to be useful for achieving a better understanding of tool infrastructure choices [9], and most importantly, provided a foundation for deep modeling, i.e., the idea of explicitly using multiple ontological classification levels for domain modeling. This approach helped shift the focus from metamodeling as a tool building technique to a

user-centered, ontological modeling paradigm [8]. Often found in combination with various forms of deep characterization [6], dual classification has therefore become the foundation for a number of research tools [22,32,14,4,21], and a growing research community [1].

In this regard, dual classification (i.e., the distinction between ontological and linguistic classification and their combined usage), has been a success. However, while the distinction between the two classification flavors is straightforward in certain architectures and application contexts [7], recognizing the two flavors and fully utilizing their strengths can be challenging in less clear-cut contexts. For instance, at first sight there does not appear to be a difference between a linguistically defined domain-specific language and an ontological multi-level model for the same domain. Some tools hence allow, if not promote, the use of ontological classification levels for doing what would widely be regarded as language engineering [5,23] even though such practice seems at odds with the ontological versus linguistic dichotomy. Such apparent interchangeability of linguistic and ontological classification makes it very difficult to judge which form of classification is, or should be, used for particular purposes, and ultimately challenges the foundations of the distinction.

In this paper we first briefly summarize the existing main expositions of ontological and linguistic classification (Sect. 2) and then elaborate the previously alluded to points of confusion (Sect. 3). Subsequently we present an expanded explanation of the distinction (Sect. 4) to then show how it can resolve all points of confusion (Sect. 5). We conclude by arguing that a proper use of both ontological and linguistic classification will be pivotal in addressing modern modeling challenges (Sect. 6).

2 Background

Fig. 1 illustrates a classic, clear-cut application of dual classification in the OCA [7]. A linguistic type model comprising the linguistic types (in the right-hand side level labeled "Linguistic Types") plays the role of a traditional language definition which controls the form of entities and their relationships in user models (in the middle-column "Model ..." levels). In contrast, the ontological types in the user type model level (middle-top "Model Types" level) represent domain classifiers, such as the Platonic idea of "Track Piece" (light bulb in the "Universe of Discourse" (UoD)). Classification relationships (labeled "ontological") between elements in adjacent user model levels represent respective classification relationships in the UoD.

Existing descriptions of the dual classification approach referred to the linguistic types as controlling "form" and the ontological types as controlling "content" [9]. Furthermore, linguistic types (such as Object) directly classify elements of language usage (such as main47), whereas ontological types (such as TrackPiece) only classify elements of language usage (such as main47) via proxy with respect to the UoD, meaning that ontological classification relationships are always motivated by respective classification in the UoD [20]. Linguistic types have therefore been characterized as giving rise to a notation / language whereas ontological types have been understood as reflecting classifiers in the domain (which may or may not exist and which may or may not have types themselves, depending on the domain).

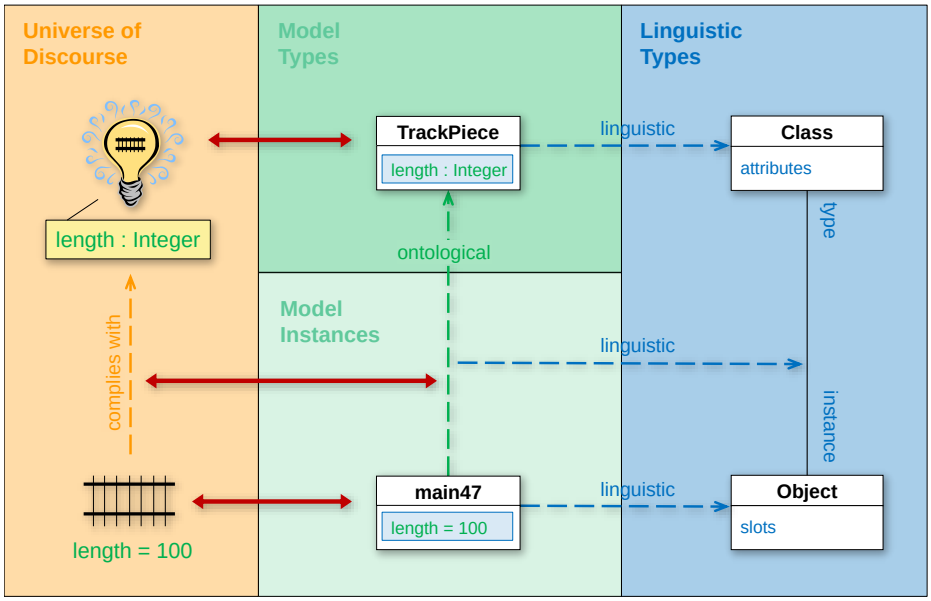

Fig. 1. Classic Dual Classification Example

We may observe that linguistic classification, as described above, has a long tradition in computer science. A classic language grammar can be regarded as linguistically classifying the allowed sentences of a language [19] and most so-called "metamodels" [28] are linguistic type models of the models they support the generation of [20]. Fig. 1 is not an attempt to accurately reflect part of the UML "metamodel" but intentionally uses a simplified approach to illustrate that TrackPiece and main47 can be regarded as modeling elements created from linguistic types Class and Object respectively. Such language definitions may incorporate well-formedness constraints that go beyond simple syntactic construction rules (static semantics [16]), but typically defer the definition of the semantics of a language (dynamic semantics [16]) to a separate transformation (Kermeta being one of the notable exceptions [27]).

We may further observe that ontological classification is intended to accurately capture the relationship between the meanings of a user-created type (here a UML class TrackPiece) and a user-created instance (here a UML object main47). The user-created type does not linguistically classify the user-created instance – i.e., it does not give the latter the ability to have a name, slots, and links – but rather just constrains the compliance of the content of the main47 object to the content of the TrackPiece class.

When the distinction between the two classification flavors is described as above it seems that they form a true dichotomy and the task of telling them apart is a trivial one. However, in the next section we will enumerate several situations which seem to challenge this assumption in order to identify points of confusion that may easily occur when applying dual classification.

3 Points of Confusion

The apparent blurring of the distinction between ontological and linguistic classification appears in scenarios that differ from the use of multi-level domain models to describe naturally occurring classification hierarchies, i.e., the scenario typically used to explain dual classification. In the following we will consider three such scenarios of particular significance to modelers:

1. Domain-Specific Languages.
2. Classic Language Engineering.
3. Dichotomy-Ambivalent Modeling.

3.1 Domain-Specific Languages

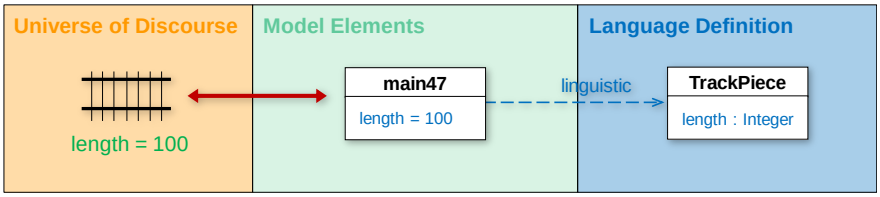

Fig. 2. A Domain-Specific Modeling Language Fragment for Train Control

In Fig. 2 we use the standard OCA coloring of levels to illustrate a case when TrackPiece is used as a linguistic type. Such a scenario occurs when a language engineer uses a metacase tool like AtoMPM [30] or just a classic textual grammar approach [19] to define a domain-specific language which aims at specifically representing elements of interest to the language user, in this example a language for train control. We are not excluding the possibility that the language engineer may also associate a domain concept "Track Piece" with the type TrackPiece they are including in their language definition, but the tool choice and the modeler's primary intent – to create signs / tokens such as main47 that are devoid of meaning unless a semantics is associated with them via a transformation – technically makes TrackPiece a linguistic type (cf. Sect. 2).

However, this interpretation of TrackPiece creates a tension with the prior understanding of TrackPiece as being an ontological type (cf. Fig. 1). The TrackPiece types in Figs. 1 & 2 are indistinguishable from each other, including the choice of attributes. Apparently the choice of a domain-specific language, rather than a general-purpose language that contains more generic types such as Object, made the previously distinct difference between ontological and linguistic classification dissolve.

This raises the question as to whether ontological classification is just a way of introducing domain-specificity into general purpose languages and, hence, whether it is then worth maintaining a dual classification scheme. At the very least the examples shown in Figs. 1 & 2 illustrate that a modeler may find it hard to ascertain whether TrackPiece should be regarded as an ontological or a linguistic type.

3.2 Classic Language Engineering

Fig. 3 depicts the case of using ontological levels (i.e., "Model Types" and "Model Instances") to perform classic language engineering, i.e., to define a language (here, in level "Model Types") to be used for some purpose (here in level "Model Instances", to represent a track piece). Note that in this scenario the purpose of Object is not to represent a domain concept but merely to create tokens such as main47 so that the latter can subsequently be used for purposes like analysis, simulation, and code generation. Therefore we did not associate the usual light bulb with Object but just an extension of all notational elements classified by Object. This, however, can be regarded as shortcut for a light bulb concept in the semantic domain of "language semantics" that has the set shown in Fig. 3 as its extension. This approach is consistent with the traditional association of a so-called "extensional semantics" for types like Object.

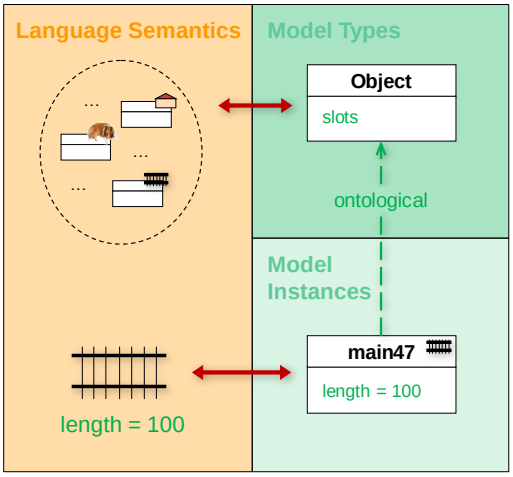

Fig. 3. Defining Notation within Ontological Levels

Tools like Melanee and MetaDepth have been shown to be usable for language engineering purposes [5,23], so it seems that Fig. 3 visualises the corresponding scenario of using ontological classification for what appears to be linguistic control.

Note that element main47 is presented using a user-friendly concrete syntax. Instead of showing the underlying representation – a slot list containing an entry that has a length name and a 100 value – main47 is presented in a manner that focuses on the content rather than the representation. Such presentation choices can go as far as rendering main47 as an icon that looks like a track piece [5]. The availability of presentation alternatives contributes to blurring the distinction between ontological classification and linguistic classification because it makes it appear that the former can now be perfectly used in place of the latter in order to perform language engineering.

However, if the use of ontological classification is not in conflict with such examples then how is it possible to determine whether a type like Object truly is a linguistic classifier or an ontological classifier?

3.3 Dichotomy-Ambivalent Modeling

The scenario shown in Fig. 4 is meant to show a multi-level model whose interpretation is ambiguous. On the one hand, the model could be read as a domain-model representing agent activities, concepts that govern those activities, and meta-concepts that govern the latter. In this case, the classification relationships between levels should be characterized as "ontological" (cf. Sect. 2).

On the other hand, the model could be read as an example of two-tier, classic language engineering. In the example shown, a process definition language is defined at the "Model Types" level and is itself the result of using a process metamodeling language. The fact that the model uses deep characterization (a potency-two attribute duration) does not rule out a language definition scenario, but rather illustrates how deep characterization can also be useful when defining (families of) languages. In any event, as the intention in language engineering is to control form, the classification relationships between the levels in Fig. 4 therefore appear to be best characterized as "linguistic".

Both of the aforementioned interpretations of Fig. 4 appear to be equally valid depending on perspective and purpose. This implies that even if one interpretation was intended at the time of creation of the model, re-purposing it for the opposing interpretation seems to be seamlessly supported. Hence, it could be argued that tools like Melanee or MetaDepth that are regularly used for defining languages as well as for domain modeling [5,23] could be regarded as not only supporting a dual purpose but, beyond that, enabling modelers to be ambivalent about their actual purpose, thus freeing the modeler from difficult deliberations. Arguably,

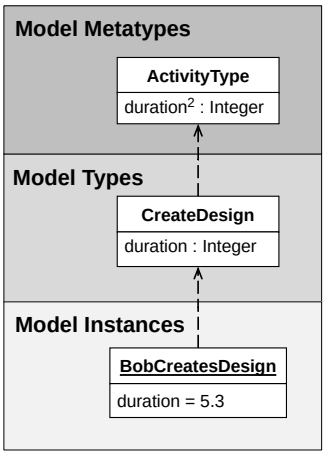

Fig. 4. Dichotomy-Ambivalence

- the same classification compliance rules can be used for both classification flavors,
- user interactions with classifiers are the same regardless of their flavor, and
- types like ActivityType apparently can be equally given an ontological as well as a linguistic reading.

Therefore, the questions arise as to

1. how one can claim that the classification flavors form a dichotomy, and
2. why one should burden users of multi-level tools with difficult deliberations about which classification principle they intend, if ambivalence even seems preferable?

That said, there is of course still a fundamental question of whether the use of ontological levels for defining languages is in accordance with the principles of dual classification (cf. Sect. 2) and, if not, whether that suggests that the principles of dual classification are a hindrance to optimal modeling pragmatics.

Summarizing, all three scenarios presented in this section strongly suggest that ontological and linguistic classification do not appear to form a long-implied "black and white" dichotomy. If types like TrackPiece and Object can interchangeably appear in both ontological and linguistic type levels and it seems best to not assign a flavor to types like "Process Type" then on what basis can anyone decide which classification flavor a type should have?

In order to answer this question in the next section, we describe the basis for the dual classification principle at a deeper level.

4 Illuminating Dual Classification

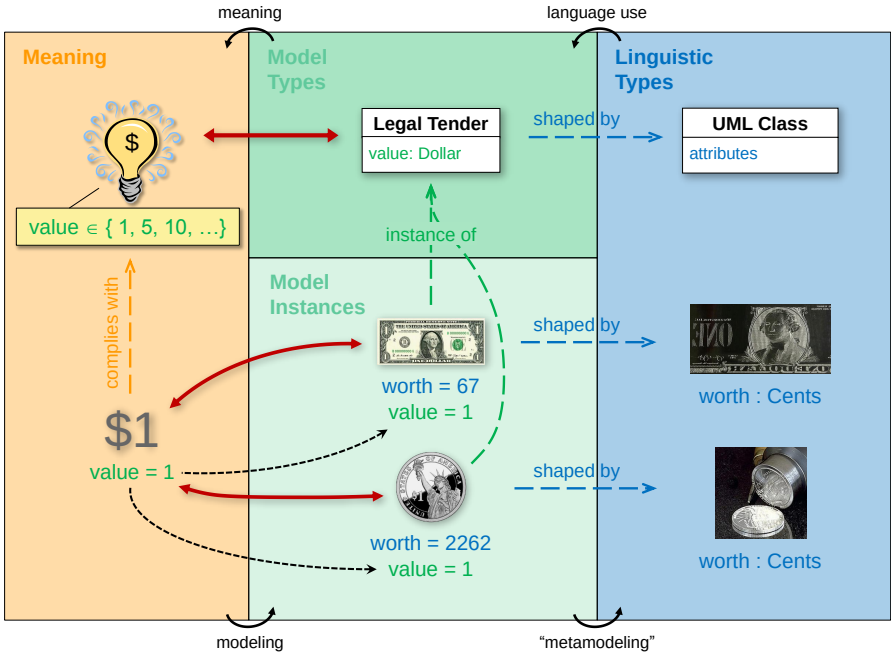

Fig. 5. Dual Classification

Fig. 5 attempts to shed more light on the distinction between linguistic and ontological classification by

- using an example that better highlights the fundamental differences, and
- illustrating a differentiating aspect that publications on the OCA have hitherto neglected.

We deliberately presented the model in Fig. 5 in a manner that allows two readings:

1. a real world scenario in which the four modeling elements dollar bill, coin, banknote mold, and coin mold can be regarded as real-world items.
2. a multi-language model that uses a domain-specific presentation for some of its modeling elements.

The idea behind the first "real world" reading of Fig. 5 is to view items of legal tender, such as a dollar bill or a dollar coin, as being formed by bill printing and coin minting molds respectively. The formed tokens (bills and coins) are then assigned "meaning", in this example their purchasing power. Note that in the real world dollar bills and coins indeed play the role of models, i.e., they are placeholders for their meaning. Purchasing power is referred to as "value" in Fig. 5 and amounts to "$1" for both bill and coin. This *value* of a legal tender item is distinguished from its *material worth*. Typically the material worth is lower (e.g., in case of the dollar bill) but it can also be higher (e.g., in the case of special collector variants of coins).

The intention behind allowing the first reading is to make it unequivocally clear that

- linguistic types can be regarded as molds. They are used in a constructive mode in the vast majority of cases to coin model elements. They simply produce tokens which are to be interpreted in a second step. The tokens have no intrinsic meaning and may have rather different meanings depending on the context. For instance, the four characters "GIFT" may mean "present" (in English) or "poison" (in German).
- generated model elements are signs / tokens which have their own intrinsic properties, independently of their meaning. In the example of Fig. 5 the materials used for the bank note are assumed to be worth 67 cents whereas the dollar coin is a silver seated liberty dollar whose melt value is $22.62. We deliberately chose an example for Fig. 5 in which the meaning of "value" is overloaded in the sense that it could apply (linguistically) to the value of the model element itself, or (ontologically) to the value assigned to the model element via an interpretation. This resolvable overloading illustrates that one must be careful to identify the subject, i.e., either the model element itself or its meaning, when attributing properties.
- generated model elements may have various meanings. In descriptive models they represent elements in the UoD but they can also have a prescriptive role, e.g., prescribing a system to be built, or simply be assigned some semantics, i.e., execution semantics. Sometimes such additional semantics are referred to as interpretations [29]. In the example in Fig. 5, the ontological interpretation of the items of legal tender is an abstract "$1" concept that only exists due to the notion of legal tender, i.e., nowadays "fiat money".
- the semantic domain of a model can reasonably be thought of containing (Platonic) ideas [7]. In the example we again use a light bulb to denote the logical idea which specifies the requirements on legal tender. Such ideas are represented by model elements at the "Model Types" level. Note that they do not specify properties of model elements, i.e., in the example LegalTender neither characterizes the dollar bill nor the dollar coin. In particular, LegalTender is not a generalization of all model element types that characterize legal tender tokens. Rather LegalTender characterizes the abstract money concept of "$1" and, of course, other amounts.

The above elaborations help to re-iterate the fact that ontological types do not directly characterize model instances. Ontological types rather represent ideas which in turn characterize instances, with the latter being represented by model instances. Linguistic types, on the other hand, directly describe properties of the tokens they produce. For example in "Love is a four-letter word" the predicate "four-letter word" applies to the word "love" itself, i.e., is a linguistic characterization. From an ontological viewpoint, the classification of "love" should be "Love is an emotion", i.e., refer to the meaning of the word "love".

With the above in mind, we may now observe that Fig. 5 illustrates an aspect of ontological classification that has so far not been mentioned in previous publications on the OCA:

- Ontological classification does not require literal conformance, i.e., in contrast to linguistic types, ontological types do not have to stipulate syntactic compliance.

In the example of Fig. 5, the seated liberty dollar absolutely must have certain physical properties, otherwise it could not be considered to be a linguistic instance of the minting mold that coined it. As the minting mold imprints all its features on all coins, they are all guaranteed to have the respective features. In the example we are assuming that the same materials will always be used in production hence every coin will feature the same material worth.

In contrast, the ontological type LegalTender specifies a requirement – i.e., for all instances to have a certain purchasing power – that is not directly expressed in its instances. None of the LegalTender instances directly carry a value feature. Whether or not they have a value is determined by looking up what they represent in the semantic domain. Only through referencing the meaning of model instances do we obtain the knowledge that both dollar bill and coin are instances of LegalTender with the value "$1" (hence the arrows from "$1" to the "value = 1"-properties in Fig. 5).

The fact that legal tender items have their intended value printed on them should not be mistaken with an expression of "meaning" as a physical property. For instance, if dollar bills were taken out of circulation then they would still claim a nominal face value of "$1" but their meaning would be "$0".

In the light of the above, we can therefore confirm that ontological instantiation can be regarded as semantically-founded and does not require literal compliance between model instances and their model types. We may thus alternatively refer to ontological classification as "semantic classification", whereas linguistic classification may be referred to as "syntactic classification". A linguistic type should be thought of syntactically classifying tokens (which may be given meaning in a subsequent step) whereas an ontological type should be thought of as semantically classifying tokens by representing a domain concept which in turn has domain instances which are represented by said tokens.

In practice, it makes sense for most ontological (semantic) classification relationships to rely on syntactic conformance as well, i.e., be no more flexible than linguistic classification. A syntactic conformance check is trivial to implement while a true semantic check would require an explicit representation of a semantic domain, the definition of a corresponding mapping, and the definition of a semantic check within the semantic domain. This significant difference in complexity explains why simple syntactic checking is almost universally accepted as a shortcut for semantic checking. Arguably, however, some languages like Eiffel [25] and JML [24], attempt to approximate a semantic check for objects by allowing the specification of pre- and post-conditions, albeit only in terms of a testing semantics. This latter limitation of ambition highlights another problem with a full semantic check: in general, its computation may be intractable or even undecidable.

5 Points of Confusion Clarified

Equipped with the above clarifications, we are now in a position to revisit the points of confusion around the dual classification principle identified in Sect. 3.

5.1 Ontological Types used for Linguistic Classification

The first challenge we elaborated upon in Sect. 3 stemmed from the fact that any domain model involving types and instances can be regarded as a (domain-specific) language definition (model types) with its corresponding language use (model instances).

However, with the function of linguistic classifiers confirmed as merely producing tokens that do not carry any inherent meaning within themselves, it becomes clear that even though types in a domain-specific language definition could have the appearance of ontological types, they do not in any way fulfill the same function.

The linguistic TrackPiece type with its attribute "length : Integer" in Fig. 2 only creates a token (placeholder) that is able to capture a value for the key "length". The ontological type TrackPiece in Fig. 1, on the other hand, denotes the existence of the Platonic idea "Track Piece" as part of a railway system where the "length" of a piece has implications for the trains that run on it, giving rise to pieces that may or may not allow collisions of trains, etc. While the actual semantics associated with the ontological TrackPiece and its main47 instance may be rather simple or may even not have a representation at all, at least in terms of potential the ontological type TrackPiece signifies something entirely different to the linguistic type TrackPiece (cf. the "value" discussion in Sect. 4).

More specifically, while the two occurrences of TrackPiece in Figs. 1 & 2 look identical and interchangeable, this observation only holds with respect to their form outside a particular context. Just as a UML class diagram may be read as a type model (e.g., with its types classifying elements in the UoD) or as a token model (e.g., with its types being tokens which represent respective Java classes) [20], it is possible to read one and the same TrackPiece type as a linguistic type or as an ontological type.

This room for interpretation, however, must not be confused with arbitrariness or a fuzzy demarcation line. Just as with the type model versus token model analogy, it is not possible to ascertain the nature of a type's classification principle from the type alone. In the absence of any knowledge regarding the role the type is playing, it is not possible to make any statement about its function and/or nature. However, once the role is known, it is no longer possible to mistake one role with the other.

As a result, arguably the choice of name for the linguistic TrackPiece type is a poor one. After all, the type actually classifies model elements, i.e., tokens, as opposed to track pieces themselves. Strictly speaking, the appropriate name for the linguistic TrackPiece type should be "TrackPieceToken" (or similar).

Note that in contrast the name for the ontological type TrackPiece should not be "TrackPieceObject" (or similar). In the ontological dimension the intent is to actually classify the domain instances themselves. Model elements such as main47 (referred to as "instance specifications" in the UML) represent domain instances but whenever they are referenced, e.g., as instances of TrackPiece, one intends to refer to their meaning, i.e., the domain instances themselves. The importance of understanding the different functions of linguistic versus ontological types, and hence the significance of proper naming, can be illustrated by analyzing what the respective types fix and what they leave open. Fig. 5 illustrates a scenario where two different tokens (coin and bill) have the same meaning, i.e., could be regarded as being synonyms. In this example, linguistic diversity is supported but semantic ambiguity is ruled out.

However, there are also homonyms, i.e., signs that are indistinguishable from each other but have different meanings. For example the sentence "*I seem to be having tremendous difficulty with my lifestyle*" has only one linguistic type (e.g., "Sentence") but depending on its ontological type (e.g., "Casually_Muttered_Phrase" versus "Dreadful_Insult_In_The_Vl'Hurg_Tongue"), it could either represent a personal self-reflection or an insult that leads to the decimation of an entire galaxy [2]. It therefore becomes obvious that knowing main47's linguistic type amounts to entirely different knowledge compared to knowing its ontological type, even though the two can seem indistinguishable on the surface.

Despite their arguably somewhat misleading naming choices (i.e. using "TrackPiece" rather than "TrackPieceToken"), classic language engineers are obviously aware that their types only define a notation, rather than capture semantic properties. After all, they use the term "metamodel" whenever they use a linguistic type model to define the syntax of a language. In contrast, a regular UML modeler would not refer to a UML class diagram which only contains simple types that represent domain concepts as a "metamodel", even though the class diagram could be regarded as a model of other models, i.e., object diagrams.

As mentioned before, we are not excluding the possibility that a language engineer may also associate a domain concept "Track Piece" with their linguistic type, thus giving more credence to their naming choice. However, as the above analysis shows, it is important to keep the two different purposes apart. Unconsciously confounding them is akin to failing to acknowledge the difference between the properties of real world elements and the properties of model elements that model them [17].

5.2 Linguistic Types used in Ontological Levels

In Sect. 3 we observed an apparent conflict due to the fact that it seemed possible to view a linguistic type like Object as an ontological type (cf. Fig. 3). Closer scrutiny reveals that two ingredients are necessary for this apparent conflict to arise:

1. the ability to choose "language engineering" as the semantic domain, and
2. the possibility to reinterpret a classification relationship.

Ontological classification between model elements mirrors logical instantiation in the domain, so when one chooses a domain in which language elements playing the role of instances are classified by language elements playing the role of types then the respective classification relationships give rise to respective ontological classification. In short, a notation and its definition can be given a structural semantics which in turn gives rise to semantic classification between the notation and its definition. This first ingredient therefore stems from the fact that any language definition combined with its corresponding language use can be regarded as a domain model with types (the language definition) and instances (the language use). In other words, a "linguistic meaning" is one of the many meanings ontological classification can embody.

Yet, this does not constitute any conflicts regarding the nature of a classifier. The ontological classifier Object in Fig. 3 has a linguistic purpose, i.e., to control the form (not the meaning) of main47. However, it achieves this purpose through ontological classification, i.e., by representing the Platonic concept of a token type (here Object).

There are two options for making this token type control instances (e.g., main47): First, the semantic domain of the token type is defined to be the ontological level where the target instances (e.g., main47) reside. This would amount to hosting actual linguistic types in ontological levels as the characteristic "compliance" relationship (cf. Fig. 5) would be missing. The second option is to choose the semantic domain of the token type to be in the same "Language Semantics" domain as the token type and defining the meaning of the target instances to be that of their counterparts in the semantic domain. A semantic check within the semantic domain validates whether there is syntactic compliance and if the latter is established then it confirms the ontological "instance of" relationship between main47 and its ontological type Object. This would support a pure ontological understanding of the form control exerted by the token type. This approach could readily be supported by any tool featuring ontological levels and the ability to map their contents into a semantic domain with an associated semantic checking function.

The fact that semantic (ontological) classification can be "downgraded" to effectively fall back to a syntactic (linguistic) check as in option 2 above, makes it impossible to judge the ultimate purpose of a type by just looking at it, even when its ontological role is known. However, any ambiguity is resolved when the context is provided. With the intended universe of discourse or semantic domain known, the type's purpose will be revealed to either classify the domain instances or the model elements.

It is worth pointing out that not every ontological type can play the role of a linguistic classifier for its model instances. While the ontological type TrackPiece in Fig. 1 could indeed play the role of a linguistic classifier for main47, the ontological type LegalTender could not play the role of a linguistic classifier for a coin, as the latter does not have a physical "value" property (only a "worth" and a mapping to a semantic domain that assigns it a value). This suggests a partial litmus test for ontological types: If the conformance between model instance and its type is not literal, i.e., not a plain syntactic conformance, then the type cannot be a linguistic type.

Since ontological classification entails an inherent ambiguity regarding the ultimate purpose (in the absence of any knowledge about the intended semantic domain), it would seem advisable to use some notation to signify the purpose of ontological types (i.e. domain modeling versus notation definition), similar to a clef in musical notation which clarifies the absolute pitch of the notes that follow it.

5.3 Postponing Role Assignments

The conclusions from section Sect. 4 and the previous analyses established that a dual interpretation of types is possible but that the respective meanings associated with the different roles are fundamentally different. Of course, this has implications for the idea of a perspective-based interpretation of types and/or the flexible re-purposing of types in approaches/tools that aim to allow users to be ambivalent about the type roles.

The premise that both ontological and linguistic classification can be supported by a tool assuming a single classification principle is correct in the sense that the types as such do not imply a commitment. Even if such a tool essentially only supports structural control over instances then it will obviously support language engineering, as well as domain modeling. However, there are disadvantages to such an approach:

Lack of Semantic Typing True semantic checking which involves transformations into the semantic domain and a subsequent check within the semantic domain is not supported. That leaves modelers with the limited expressiveness and flexibility of syntactic typing, denying them the additional abstraction that semantic typing affords.

Only Simple Language Support Ontological classification hierarchies are linear by nature. If language engineering is restricted to a linear hierarchy, however, it is difficult to cleanly support languages with a built-in notion of classification. Enabling a nesting of levels to accommodate language engineering would also be at odds with the premise that no commitment to a classification role is ever required since nesting would not make sense for an ontological interpretation.

Ambiguity Considered Harmful Arguably, it makes sense for a modeler to be conscious about what classification flavor they have in mind. The choice of features and their names, for example, can depend on whether one intends a semantic or a syntactic commitment (cf. Sect. 4, regarding the overloading of "value" for coins). Also, in the case of a semantic commitment description logics could be used to capture semantic knowledge in the domain whereas in the case of a syntactic classification only, simple attributes are sufficient as a specification. Finally, if a modeler is not clear about the intended role, they may mix linguistic and ontological roles in a single model without realizing the inconsistencies. In one type an attribute may be labeled "nameString" (indicating a linguistic intent) whereas an associated type could use "name" (indicating an ontological intent).

The first issue from above could be addressed by choosing an ontological interpretation as the default and viewing applications of language engineering as ontological modeling with respective structural checks performed in the semantic domain. The second issue, however, points out a real limitation of tools supporting linear classification levels with respect to defining languages that feature a notion of instantiation. While it is possible to model such instantiation relationships, the tool would not be able to recognize and support their significance. The last point suggests that future work should clarify which kinds of ambivalence are welcomed as supporting re-purposing and which may be considered harmful as they mask fundamental differences.

6 Conclusion

Linguistic classification has an undisputed role in computer science as the basis for formalization and classic language engineering. While the recognition of ontological classification has helped to spawn a research field, its previous expositions have also created misunderstandings [10] and made it difficult to distinguish it from linguistic classification in certain scenarios (cf. Sect. 3).

In order to clarify the role of ontological classification, in this paper we identified critical differences between linguistic and ontological classification that have not been highlighted before. We observed that

– "*semantic classification*" could be an alternative name for ontological classification as it emphasizes the inherent reference to a UoD or a semantic domain.

- ontological classification does not require literal conformance as it captures semantic properties of subject instances, as opposed to creating carriers for semantics.
- the use of the name "linguistic classification" should not be construed to imply that all language definition must exclusively occur through linguistic classification.

Earlier publications on the ontological versus linguistic dichotomy only dealt with straightforward scenarios and hence did not highlight the above aspects. In this paper we furthermore made the key observation that in order to avoid confusion one must reject the assumption that a type is intrinsically either an ontological or a linguistic type. We clarified that one and the same type may play an ontological role in one context and a linguistic role in another context (cf. Sect. 4). We thus emphasized that a type's purpose in a particular context is important to understand its role and that the dichotomy therefore does not apply to types themselves, but to the roles they play.

Yet, even if an ontological role is confirmed, e.g. by applying respective litmus tests (cf. Sect. 5.2), the intended use of the type may not be entirely clear. In Sect. 5.2 we noticed that this stems from the fact that a semantic test can boil down to a structural conformance check and that the respective ontological classification can hence be indistinguishable from linguistic classification, in terms of its effect and in the absence of knowledge about the semantic domain.

Sect. 3.3 made the case that such ambiguity could be the basis for an approach that promotes dichotomy ambivalence as a feature. However, we also pointed out a list of limitations associated with linear hierarchies built on this principle (cf. Sect. 5.3). We believe future work should provide a comprehensive analysis of the trade-offs involved in using the "ambivalent classification" approach. On the one hand it appears to liberate modelers from potentially difficult deliberations, but on the other hand modeler obliviousness may cause inconsistencies and even inappropriate modeling (cf. Sect. 5.1). There is no immediate resolution to this issue since – as we mentioned in Sect. 3.1 – the definition of a notation need not always be in conflict with simultaneously capturing domain semantics. Hence, the options of banning ontological classification from only exerting form control, explicitly distinguishing within ontological hierarchies between domain semantics versus form control, or promoting ambivalence or even agnosticism should be evaluated in future work.

Undoubtedly, however, we expect ontological classification to play an integral role in the future of modeling when used with the expanded interpretation we have offered in this paper. First, by exploiting the liberation from syntactic conformance it is possible to accommodate more flexible classification relationships based on meaning rather than syntax. For example, immaterial differences, such as different naming choices like "diameter" versus "width", do not prevent instances from being recognized as belonging to the same category (e.g., "Shape"). Such emphasis on the meaning rather than the structure of data is the underpinning for the "Semantic Web" and its associated "Web Ontology Language".

Second, ontological classification provides a new means of injecting semantics into modeling. In contrast to Barroca et al. [11] who use ontological types to support the reuse of property definitions (e.g. "liveliness", "safety", etc.) that are otherwise often captured with additional property specification languages [26], we suggest that ontological types should also incorporate domain semantics for natural types such as "Special-

istWorker", etc. In other words while we support the use of ontological (property) types to represent so-called "appredicators", we believe ontological types will also prove to be very useful for representing so-called "(eigen-)predicators".

We thus advocate ontological types as a bridge [18] between the semantically-oriented world of ontology engineering [12] and the syntactically-oriented word of classic language engineering [19]. Enhancing traditional language definitions with semantic properties that advanced tools will be able to validate through checks ranging from simple conformance checking involving name mapping, through simulations, to model-checking and automated proofs, will make modeling more meaningful than it has been in the context of software engineering. The systems modeling community has a longer tradition of associating semantics to languages [26] but even for this community the use of user-defined semantics represented with ontological types is a novel concept.

We support the view taken by Vangheluwe et al. that tackling the challenges involved in modeling complex systems, such as cyber-physical systems [15], requires the use of multiple languages/formalisms and the incorporation of semantics [31,13]. Introducing semantic properties, validating them, and demanding their preservation in modeling transformations will be a crucial tool to master the complexity of modeling and generating contemporary systems.

In this paper we have not attempted to identify the optimal architecture for supporting multiple languages along with a semantic perspective on the UoD. However, we hope that our clarification of the distinction between ontological classification and linguistic classification will contribute towards identifying useful roles for ontological classification in the context of classic language engineering. It is in this light that we emphasize there are no grounds for the assumption that claiming a difference between ontological and linguistic classification creates more problems than it solves. On the contrary, we believe ontological classification should be given more consideration in classic language engineering than it has been given to date.

References

1. MULTI-LEVEL MODELING WIKI (2014), http://homepages.ecs.vuw.ac.nz/Groups/MultiLevelModeling/
2. Adams, D.: The Hitchhiker's Guide to the Galaxy. Del Rey (September 1995)
3. Atkinson, C.: Meta-modeling for distributed object environments. In: Enterprise Distributed Object Computing. pp. 90–101. IEEE Computer Society (Oct 1997)
4. Atkinson, C., Gerbig, R.: Melanie: Multi-level modeling and ontology engineering environment. In: Proceedings of Modeling Wizards'12. ACM (2012)
5. Atkinson, C., Gerbig, R., Kennel, B.: Symbiotic general-purpose and domain-specific languages. In: Proceedings of the 34th International Conference on Software Engineering. pp. 1269–1272. ICSE '12, IEEE Press, Zurich, Switzerland (2012)
6. Atkinson, C., Kühne, T.: The essence of multilevel metamodeling. In: Gogolla, M., Kobryn, C. (eds.) Proceedings of the 4^{th} International Conference on the UML 2000, Toronto, Canada. pp. 19–33. LNCS 2185, Springer Verlag (Oct 2001)
7. Atkinson, C., Kühne, T.: Model-driven development: A metamodeling foundation. IEEE Software 20(5), 36–41 (Sep 2003)
8. Atkinson, C., Kühne, T.: Rearchitecting the UML infrastructure. ACM Transactions on Modeling and Computer Simulation 12(4), 290–321 (Oct 2003)

9. Atkinson, C., Kühne, T.: Concepts for comparing modeling tool architectures. In: Briand, L. (ed.) Proceedings of the ACM/IEEE 8^{th} MODELS. pp. 398–413. Springer Verlag (2005)
10. Atkinson, C., Kühne, T.: In defence of deep modelling. Information and Software Technology 64, 36–51 (August 2015)
11. Barroca, B., Kühne, T., Vangheluwe, H.: Integrating language and ontology engineering. In: Proceedings of the 8th Workshop on Multi-Paradigm Modeling. vol. Vol-1237, pp. 77–86. CEUR-Workshop Proceedings (September 2014)
12. Carvalho, V.A., Almeida, J.P.A., Fonseca, C.M., Guizzardi, G.: Extending the foundations of ontology-based conceptual modeling with a multi-level theory. In: Proceedings of ER'15. pp. 119–133. Springer (2015)
13. Combemale, B., Deantoni, J., Baudry, B., France, R., Jézéquel, J.M., Gray, J.: Globalizing Modeling Languages. Computer pp. 68–71 (Jun 2014)
14. Demuth, A., Lopez-Herrejon, R.E., Egyed, A.: Cross-layer modeler: a tool for flexible multi-level modeling with consistency checking. In: 19th Symposium on the Foundations of Software Engineering (FSE), Szeged, Hungary. pp. 452–455 (2011)
15. Derler, P., Lee, E.A., Sangiovanni-Vincentelli, A.: Modeling cyber-physical systems. Proceedings of the IEEE (special issue on CPS) 100(1), 13 – 28 (January 2012)
16. Harel, D., Rumpe, B.: Modeling languages: Syntax, semantics and all that stuff - part I: The basic stuff. Tech. Rep. MCS00-16, The Weizmann Institute of Science, Israel (Sep 2000)
17. Jackson, M.: Some basic tenets of description. SoSyM 1(1), 5–9 (2002)
18. Kappel, G., Kapsammer, E., Kargl, H., Kramler, G., Reiter, T., Retschitzegger, W., Schwinger, W., Wimmer, M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In: Proceedings of MODELS'06. LNCS, vol. 4199, pp. 528–542. Springer (2006)
19. Klint, P., Lämmel, R., Verhoef, C.: Toward an engineering discipline for grammarware. ACM Trans. Softw. Eng. Methodol. 14(3), 331–380 (Jul 2005)
20. Kühne, T.: Matters of (meta-) modeling. SoSyM 5(4), 369–385 (2006)
21. Lamo, Y., Wang, X., Mantz, F., MacCaull, W., Rutle, A.: DPF Workbench: A diagrammatic multi-layer domain specific (meta-)modelling environment. In: Lee, R.Y. (ed.) Computer and Information Science 2012, vol. 429, pp. 37–52. Springer (2012)
22. de Lara, J., Guerra, E.: Deep meta-modelling with metadepth. In: Proceedings of TOOLS (48). pp. 1–20 (2010)
23. Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. Softw. Syst. Model. 14(1), 429–459 (Feb 2015)
24. Leavens, G.T., Baker, A.L.: Enhancing the pre- and postcondition technique for more expressive specifications. In: Wing, J.M., Woodcock, J., Davies, J. (eds.) World Congress on Formal Methods. LNCS, vol. 1709, pp. 1087–1106. Springer (1999)
25. Meyer, B.: Object-Oriented Software Construction. Prentice Hall, 2nd edn. (1997)
26. Meyers, B., Wimmer, M., Vangheluwe, H., Denil, J.: Towards domain-specific property languages: The promobox approach. In: Proceedings of DSM'13. pp. 39–44. ACM (2013)
27. Muller, P.A., Fleurey, F., Jézéquel, J.M.: Weaving executability into object-oriented meta-languages. In: Proceedings of MODELS'05. pp. 264–278. Springer-Verlag (2005)
28. OMG: Unified Modeling Language Infrastructure Specification, Version 2.0 (2004)
29. Seidewitz, E.: What models mean. IEEE Software 20(5), 26–32 (Sep 2003)
30. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Mierlo, S.V., Ergin, H.: AToMPM: A web-based modeling environment. In: Joint Proceedings of MODELS'13 and ACM Student Research Competition. pp. 21–25 (2013)
31. Vangheluwe, H., de Lara, J., Mosterman, P.: An introduction to multi-paradigm modelling and simulation. In: Proceedings of the AIS'2002 Conference, Portugal. pp. 9–20 (2002)
32. Volz, B., Jablonski, S.: OMME – A flexible modeling environment. In: Proceedings of SPLASH Workshop on Flexible Modeling Tools (FlexiTools) (2010)