A Note on Properties in Multi-Level Modeling

João Paulo A. Almeida*, Victorio A. Carvalho†,
Claudenir M. Fonseca‡, Giancarlo Guizzardi‡§
*Ontology & Conceptual Modeling Research Group (NEMO),
Federal University of Espírito Santo (UFES), Vitória, Brazil
†Federal Institute of Espírito Santo (IFES), Colatina, Brazil
‡Conceptual and Cognitive Modeling Research Group (CORE),
Free University of Bozen-Bolzano, Bolzano, Italy
§Services & Cybersecurity Group,
University of Twente, The Netherlands
jpalmeida@ieee.org, victorio@ifes.edu.br,
cmoraisfonseca@unibz.it, giancarlo.guizzardi@unibz.it

***Abstract*—Given their ubiquity in conceptual modeling languages, it is no surprise that properties have been subject to attention and specialized support in multi-level modeling approaches (including mechanisms such as deep characterization). This paper examines consequences of a typology for properties of high-order types that distinguishes them into: direct, resultant and regularity properties. We discuss several implications of the proposed classification considering a number of aspects of multi-level modeling including: specialization between high-order types, applicability to powertype variants, property change, and potency.**

I. Introduction

Conceptual models capture invariant aspects of entities in a domain of interest. They are often defined through the identification of the relevant types (or "classes") of entities that are admitted in a domain, along with the relations that these entities may have and the features they may exhibit. Relations and features are specified with constructs such as "associations", "properties" and "attributes" (in UML [15]) or "(object and data) properties" (in OWL [16]).

Given their ubiquity in conceptual modeling languages, it is no surprise that relations and features alike have been subject to attention and specialized support in multi-level modeling approaches. In fact, a key feature of multi-level modeling techniques is the capacity to consider a class as an instance of a metaclass and possibly assign values to properties defined at the metaclass level. For example, if we consider `Car Model` as a metaclass, with instances such as `Ferrari F40` and `Volvo S60`, then a property `model designer` may be defined for `Car Model` and attributed to `Ferrari F40` and `Volvo S60`.

Beyond this basic support for assigning values to properties of metaclasses, some multi-level modeling techniques identify the opportunity to establish correlations or connections between features of classes at different levels of classification, what has been referred to as "deep characterization" [4].

Deep characterization is an important mechanism for multi-level modeling as it enables a modeler to capture invariants at a certain level that influence not only the level immediately below (what would constitute a "shallow" mechanism) but also other subsequent levels deeper in the classification level scheme (hence the term "deep" characterization). For example, if we conceive `Animal Species` as a metaclass, whose instances include the classes `Platypus`, `Dog`, and `Lion`, then whether the lion `Cecil` has the property of being warmblooded is in fact determined by a property of its classifying species [9] (i.e., determined by `Lion` being a species of warmblooded animals). If we consider `Mobile Phone Model` as a metaclass, whose instances include `iPhone 12` and `Samsung S21`, we may include screen size as a property of `Mobile Phone Model`, and assign values to it for `iPhone 12` and `Samsung S21` (6.6 and 6.2 inches respectively). The screen sizes of instances of `iPhone 12` and `Samsung S21` will then follow those values.

Different multi-level modeling techniques address deep characterization in different ways. In MLT and in the MLT-based language ML2, this phenomenon is addressed through "regularity attributes" [6], [10]. In Melanie [2], they are supported through a combination of the notions of "potency" along with attribute "durability" and "mutability". In MetaDepth [7] they are also supported by potency and durability. (See [9] for a review of several MLM approaches with respect to their support for this kind of attribute.)

This short paper examines different phenomena involving properties in multi-level modeling building up on the typology for properties for metaclasses identified earlier in [10]. We discuss here some implications of that typology. In particular, we show that the different types of properties have different implications for their inheritance along specialization hierarchies, for their applicability in powertype variants, their (im)mutability and use in potency-based schemes.

This paper is further structured as follows: Section II establishes basic terminology and presents the typology of properties introduced originally in [10], Section III discusses various implications of the typology for multi-level modeling, and Section IV concludes this paper.

II. Background

A. Preliminary Considerations

Establishing uniform terminology for the phenomena we are considering here is a challenging task. We settle here to

establish some basic terminology for the purpose of this paper. We will adopt here the term "property" in alignment with the UML, which introduces it in the context of a classifier as follows (extracted from [15]): "A Property related by ownedAttribute to a Classifier [...] represents an attribute and might also represent an association end. It relates an instance of the Classifier to a value or set of values of the type of the attribute." By doing so, we can address at the same time UML's attributes and association ends. The same account is also suitable for OWL object properties (which can roughly be understood as navigable association ends) and data properties (which can roughly be understood as attributes typed with datatypes).

Further, we consider each type to be characterized by an *intension* (or *principle of application* [11]), which is used to judge whether the type applies to an entity (e.g., whether something is a `Person`, a `Dog`, a `Chair`). If the intension of a type t applies to an entity e then it is said that e is an instance of t. The set of instances of a type is called the extension of the type [12]. We admit that types may have a time-varying extension, when entities that fall under the type's intension are created and destroyed.

Finally, we classify types of a multi-level model into *orders*. The types whose instances are individuals are called *first-order types*. The types whose instances are first-order types are called *second-order types*, and so on. Second-order types (also termed metaclasses), third-order types (also termed metametaclasses), and so on, are termed collectively *high-order types*.

B. Typology for Properties

Following [10], we make use of examples in the domain of biological species to illustrate the various types of properties. The modeling of species (and animal breeds alike) is evoked as a typical example of multi-level modeling in the literature (see, e.g., "tree species" in [15] and "dog breed" in [3]). We also employ examples of product types [6].

Consider the example shown in Figure 1, using a notation inspired in UML. `Bird` is specialized in two subtypes, namely, `Blue Macaw` and `Golden Eagle`. According to this model, particular birds have a particular `birth date`, a particular `height`, and a `name`. This model uses the powertype pattern: the two subtyping relations between the latter types and `Bird` are part of a generalization set related to the powertype `Bird Species`. The relation that connects `Bird` and `Bird Species` represents instantiation, declaring that the instances of `Bird` are classified by instances of `Bird Species`. Furthermore, the powertype `Bird Species` is connected to the type `Bird` by being referred to in the generalization set specializing `Bird` and containing the subtypes `Blue Macaw` and `Golden Eagle`. Hence, `Blue Macaw` and `Golden Eagle` are instances of `Bird Species`. (`Bird Species` is a second-order type, whose instances are first-order types.) Two instances of `Bird` in this model are `Blu` (a particular `Blue Macaw`) and `Joe` (a particular `Golden Eagle`). The instances are related to their types through dashed arrows labeled `instance of`.

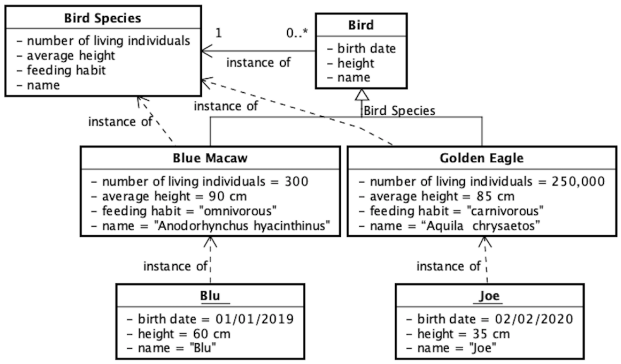

Figure 1. Bird Species example.

Note that instances of `Bird Species` provide specific values for all the general properties that characterize the type `Bird Species`. For instance, the type `Golden Eagle` may have a `number of living individuals` = 250,000, and an `average height` = 85 centimeters. Notice that these are not properties of particular birds (e.g., Joe does not have an average height, or a number of living individuals), but properties of each species of birds as a whole. Indeed, properties such as number of living individuals or average height are properties of instances of `Bird Species` that result from properties of the instances of `Bird` (e.g., the average height of a particular species such as `Golden Eagle` is derived from individual heights of particular instances of `Golden Eagle`). We term these properties *resultant properties* of the species. They are derived (or derivable) from the extension of the type (the population of birds and their properties). A fully specified resultant property includes the definition of the means for derivation, e.g., in terms of the counting of instances or any other form of aggregation of values of properties of a lower-level type.

In contrast, a property such as `feeding habit` for `Bird Species` capture regularities over the instances of a particular type. When declaring that `feeding habit` is `"carnivorous"` for `Golden Eagle`, we are capturing that all instances of that type are carnivores. To be precise, the type `Golden Eagle` is not itself a carnivore; it has the property of having instances that exhibit that property. In other words, it has the property of bestowing to all its instances a particular feeding habit[1]. We term here these properties *regularity properties*. The aforementioned property `screen size` of `Mobile Phone Model` is another example of regularity property. Regularity properties can be understood as parameters in the intension of the instances of the types that have it. Since regularity properties affect the intension of instances of a type, they can only be defined for high-order types [6] (thus neither for individuals nor for first-order types).

Finally, a property such as the species name or the year in which it was officially recognized are properties of yet a third

[1]This model reflects the assumption of "intrinsic biological essentialism" that there are some essential intrinsic properties shared by members of a species [8].

kind. For instance, being officially recognized as a species in 1760 and being named Aquila chrysaetos are properties of the type Golden Eagle and not a property that any individual instance of Golden Eagle has. We term these properties *direct properties*.

III. Implications of the Typology of Properties

A. Implications to Specialization

When a type specializes another type, all instances of the subtype are also instances of the supertype. This means that the intension of the subtype includes the conditions in the intension of the supertype; often, subtypes add further classification criteria to a supertype, restricting those instances of the supertype that fall under the subtype; in this case, specialization may be termed "proper specialization" [6].

This basic understanding of specialization allows us to draw a rather straightforward consequence for regularity properties: whenever a supertype is given a value to a regularity property—say, when we establish that Mobile Phone Model is characterized by the screenSize regularity property, and iPhone X instantiating Mobile Phone Model has screen size=5.85 inches—all its subtypes (such as Red iPhone X) impose that same value on their instances. In other words, since regularity properties establish invariant aspects of the instances of a type, and the instances of a subtype are also instances of the supertype, the invariants defined for the instances of the supertype must also be respected by instances of the subtypes.

The same cannot be said of resultant properties. Since the value of a resultant property depends on the instances of a type, and subtypes may have extensions that subset the extensions of the supertype, the value of a resultant property given for a supertype is not necessarily preserved in subtypes. For example, currently, the number of living individuals of the supertype Macaw is greater than that of Blue Macaw (as this is not the sole species of macaw with living individuals). Further, the average height of Blue Macaw differs from that of its subtypes Male Blue Macaw and Female Blue Macaw. Despite that, since a resultant property may be derived from properties that characterize the supertype (in the latter case, height that characterizes Bird), and all instances of the subtype inherit those properties, it is possible to extend the definition of the resultant property—but not its value attribution—to all subtypes of the instances of the high-order type characterized by the resultant property. In this case, Male Blue Macaw and Female Blue Macaw may be given a value for average height even though they are not instances of Bird Species. In order to explain this, it is useful for us to consider the relation of *subordination* between high-order types as discussed in MLT [6]. A high-order type is *subordinate to* another high-order type if, and only if, each of its instances specialize an instance of the superordinate high-order type [6]. For example, Bird Type by Species and Sex is subordinate to Bird Species, and hence its instances (Male Blue Macaw and Female Blue Macaw) specialize an instance of Bird Species (Blue Macaw). We can then conclude that subordinate high-order types may "inherit" resultant properties from their superordinate types.

Differently from regularity and resultant properties, direct properties are not "inherited" in any sense, as they pertain solely to the type, and not its instances. Consider that Mobile Phone Model is characterized by a launch date direct property. The fact that iPhone X has launch date 11/03/2017 does not determine the launch dates of types specializing it. In fact, the notion of a launch date for subtypes of iPhone X may not even be meaningful; consider the case of Refurbished iPhone X.

In summary, the typology of properties of high-order types we adopt here has the following consequences: (i) the values of regularity properties are preserved in specialization, (ii) resultant properties are inherited by subordinate high-order types (but their values are not preserved), and (iii) direct properties are not "inherited" in any sense.

B. Implications to Powertypes

The typology of properties of high-order types also has consequences to the variants of the powertype pattern identified in [6]. More specifically, we refer here to the variants of Cardelli [5] and Odell [14], which lead to the following relations between types [1]:

A *powertype* relation to capture the notion of powertype as defined by Cardelli [5]: a type *pt is powertype of* a (base) type t iff all instances of *pt* are specializations of t and all possible specializations of t are instances of *pt*. Powertypes in this sense are analogous to powersets[2]. The powerset of a set A is a set that includes as members all subsets of A (including A itself). As an example of Cardelli powertype consider Bird Type defined in such a way that all possible specializations of Bird (including Bird) are instances of Bird Type.

A *categorization* relation between types was defined to reflect Odell's notion of powertype [14]. Differently from Cardelli's, Odell's definition excludes the base type from the set of instances of the powertype. Further, not all specializations of the base type are required to be instances of the powertype. Odell's definition is more similar to the notion of powertype that was incorporated in the UML. There may be specializations of the base type that are not instances of the categorizing higher-order type. For example, Bird Type by Species and Sex (with instances Male Sparrow and Female Blue Macaw among many others) *categorizes* Bird. Bird Type by Species and Sex is not a (Cardelli) powertype of Bird since there are specializations of Bird that are not instances of Bird Type by Species and Sex (e.g. Blue Macaw and Golden Eagle).

From the definition of a Cardelli powertype, we can suggest that Cardelli powertypes should not be characterized by

[2]Notice that we use the term 'analogous' and not 'identical'. Powertypes are analogous to powersets in the sense that they have as instances all possible subtypes of a base type, including that base type itself. However, we do not hold an extensional view on types. In fact, we reject the idea that, in general, for a powertype t of base type t', there should be an instance of t corresponding to any set belonging to the powerset of the extension of t'.

regularity properties. This is because there are, in general, many ways to constrain the various properties of instances of a base type, and hence, these many ways cannot be covered by a single regularity property. For example, if we define that screen size is a regularity property defined in the type Mobile Phone Type it is not possible for Mobile Phone Type to be a powertype in Cardelli's sense since there are possible specializations of Mobile Phone that do not impose constraints over the screen size of their instances (e.g., 5G Phone).[3]

The same constraint does not apply to resultant properties in the case of Cardelli powertypes; as long as these are derived in terms of properties of the base type, they can in principle be defined for all subtypes of the base type.

Since Cardelli powertypes have a "formal" (rather then domain-specific) nature, the direct properties in Cardelli powertypes are restricted to general properties of types such as name, date of creation, etc.

C. Implications to the Dynamics of Properties

The typology of types also has consequences to their dynamics in time. Regularity properties are immutable in principle because, by definition, changing their value alters the identity of the type [6]. Any alleged change of the value of a regularity property is actually the creation of another type. In turn, the value of a resultant property may change as long as the extension of the type changes, or the property of instances in the extension changes. In the special case that the extension of a type is constant and the resultant property is defined in terms of immutable properties of a lower-level type, then we have an immutable resultant property. Direct properties, in turn, may be mutable (e.g., is currently in production for Mobile Phone Model) or not (e.g., launch date), in a way that does not depend on the instances of lower-level types.

D. Implications to Potency-based Approaches

Here we discuss some consequences to potency-based approaches. More specially, we select Melanie [2], [13] as a prototypical exemplar of potency-based approach given its historical role. Melanie addresses deep characterization through a combination of the notions of "clabject" and "potency" along with attribute "durability" and "mutability".

The notion of "clabject" is founded on the observation that every instantiable entity has both a type (or class) facet and an instance (or object) facet which are equally valid. For example, Golden Eagle can be considered a clabject since it has an instance facet (it is an instance of Species) and a type facet (it classifies Joe as an instance of Golden Eagle).

Each clabject has a potency assigned to it. The potency of a clabject is an integer that defines the depth to which a model element can be instantiated. When a clabject is instantiated from another clabject the potency of the created clabject is given by the original clabject potency decremented by one. Clabjects have potency equal to zero indicating they cannot be instantiated, which is the case of individual objects.

The attributes that characterize a clabject have both a "durability" and a "mutability" assigned to them. The durability defines the endurance of the attribute over the instantiation chain. It is a non-negative integer that is decremented by one when the clabject characterized by the attribute is instantiated. When durability reaches zero the instantiated clabject no longer is characterized by that attribute. The mutability of an attribute defines how often its value can be changed over the instantiation chain. Like durability, mutability is a non-negative integer that is decremented by one when the clabject is instantiated. When mutability reaches zero, the value of the attribute can no longer be changed and must be the same as in the level above. The default value for both durability and mutability is the potency value of the owning clabject.

The simplest case is that of durability and mutability equals to one. This corresponds to a shallow characterization, and can be used to represent direct or resultant properties.

More complex scenarios can be captured combining different values of durability and mutability. Consider, for example, a type Mobile Phone Model with potency 2. An attribute screen size with durability 2 and mutability 1 will be given a value at the first instantiation (e.g., stating that the iPhone X has screen size equal to 5.85 inches), and that value will determine the value of screen size for the instances of instances of Mobile Phone Model (thus, all instances of iPhone X have a screen size of 5.85 inches). In our view, this representation captures concisely the constraint relating: a property defined for the first-order type Mobile Phone (screen size), and; a regularity property of Mobile Phone Model (instances screen size). This constraint is captured as a single attribute with durability 2 and mutability 1 in a clabject with potency 2 (Mobile Phone Model).

The combination between durability and mutability greater than one may also be used to capture scenarios in which homonymous direct attributes characterize both a type and its instances. For example, consider a type Bird Species with potency 2. An attribute name with durability and mutability 2 may be defined in Bird Species to represent that instances of Bird Species have a name (e.g. "Aquila chrysaetos") and that instances of instances of Bird Species (i.e., instances of Bird) also have a name (e.g. "Pat"). Therefore, the attribute name with durability and mutability 2, conflates the direct property name of the second-order type Bird Species and the property name of the first-order type Bird. It is important to notice that there is no relation or constraint linking the values of the attribute at type level with the possible values of attributes at object level, i.e., the name of the species does not constrain the possible names of the specific birds.

A feature of potency-based approaches in general (and of Melanie specifically) is that they allow the modeler to omit the representation of base types as a way of reducing complexity. In these scenarios, the direct attributes of the base types are

[3]Regularity properties would only be admissible in Cardelli powertypes in the very special case that the regularity property may take on a value from the space of all possible constraints applicable to the base type—a sort of trivial or catch-all regulation.

represented as attributes of high-order types with durability and mutability greater than one. For example, in order to capture that a specific bird has a birth date, without the need of representing the type `Bird`, the attribute `birth date` may be defined as an attribute of `Bird Species` having both mutability and durability 2, so that instances of instances of `Bird Species` may define values for the attribute `birth date`. An undesirable effect of this approach is that it is possible to set a value to the attribute `birth date` at the species level, which is meaningless (a species does not have a birth date).

Note that we have a case of construct overload here: the language does not provide means to distinguish between (i) the cases in which attributes with durability and mutability greater than 1 are used to conflate direct properties of types in different levels (with possibly different values at various levels), and (ii) the cases in which those attributes are used to represent, in the higher-order type, properties of omitted lower-level types.

IV. Conclusions

Not all properties of high-order types behave the same. We have shown that attention to the role of properties in establishing the relation between the various levels in a multi-level model can provide us with some guidelines for their usage. We have leveraged here the typology of properties defined originally in [10], which distinguishes them into *regularity properties*, *resultant properties* and *direct properties*.

We have argued that *regularity properties* impose constraints for their instances' instances that are preserved throughout specialization hierarchies. More specifically, when a value is attributed to a regularity property for a supertype, this value is the same for all its subtypes. Further, they are immutable and not applicable to Cardelli powertypes. We have discussed that a specific type of regularity property can be captured using durability greater than one in a potency-based approach; but the same specification strategy is also used for other modeling scenarios not involving regularity (e.g., to omit base types at lower levels), which may hinder the interpretation of models using this specification strategy.

In their turn, *resultant properties* may be "inherited" throughout *subordination* hierarchies, are in general mutable and may be applied to Cardelli and Odell powertypes alike.

Finally, *direct properties* are not "inherited" in any sense; they correspond simply to properties in shallow classification schemes (or to attributes with durability and mutability equal to one in a potency-based approach; these attributes, once given a value in an instance of a clabject, no longer belong to the type facet of the clabject as discussed in [13]).

Given that identifying the type of property gives us some guidelines for their usage in a high-order type, future work could try to establish the applicability of these distinctions as constructs in multi-level modeling languages, regardless of whether they are powertype-based or potency-based. Ideally, the modeler should be able to express the type of property they indent to represent (e.g., through specialized syntactic constructs) and reap benefits of some automated support. To the best of our knowledge, the distinctions discussed here are not yet incorporated fully in the various approaches.

Acknowledgment

This study was financed in part by CAPES (grant 23038.028816/2016-41) and CNPq (grants 313687/2020-0 and 407235/2017-5). Claudenir M. Fonseca and Giancarlo Guizzardi are supported by the NeXON Project (Free University of Bozen-Bolzano).

References

[1] J. P. A. Almeida, V. Carvalho, F. Brasileiro, C. M. Fonseca, and G. Guizzardi, "Multi-Level Conceptual Modeling: Theory and Applications," in *Proc. XI Seminar on Ontology Research in Brazil*, ser. CEUR Workshop Proceedings, vol. 2228, 2018, pp. 26–41.

[2] C. Atkinson and R. Gerbig, "Melanie: Multi-level modeling and ontology engineering environment," in *Proc. 2nd Int Master Class on Model-Driven Engineering: Modeling Wizards (MW '12)*. New York, NY, USA: ACM, 2012, pp. 7:1–7:2.

[3] C. Atkinson and T. Kühne, "The Essence of Multilevel Metamodeling," in *«UML»2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools*, ser. LNCS, vol. 2185. Springer, 2001. [Online]. Available: https://doi.org/10.1007/3-540-45441-1_3

[4] C. Atkinson and T. Kühne, "Reducing accidental complexity in domain models," *Software & Systems Modeling*, vol. 7, no. 3, pp. 345–359, Jun. 2007. [Online]. Available: https://doi.org/10.1007/s10270-007-0061-0

[5] L. Cardelli, "Structural subtyping and the notion of power type," in *Proceedings of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ser. POPL '88. New York, NY, USA: ACM, 1988, pp. 70–79. [Online]. Available: http://doi.acm.org/10.1145/73560.73566

[6] V. Carvalho and J. P. A. Almeida, "Toward a well-founded theory for multi-level conceptual modeling," *Software & Systems Modeling*, pp. 1–27, 2018. [Online]. Available: https://doi.org/10.1007/s10270-016-0538-9

[7] J. de Lara and E. Guerra, "Deep meta-modelling with MetaDepth," in *Objects, Models, Components, Patterns. TOOLS 2010*, ser. LNCS, vol. 6141. Springer, 2010, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-642-13953-6_1

[8] M. Devitt, "Species have (partly) intrinsic essences," *Philosophy of Science*, vol. 77, no. 5, pp. 648–661, 2010. [Online]. Available: http://www.jstor.org/stable/10.1086/656820

[9] C. M. Fonseca, J. P. A. Almeida, G. Guizzardi, and V. Carvalho, "Multi-level conceptual modeling: Theory, language and application," *Data & Knowledge Engineering*, vol. 134, p. 101894, Jul. 2021. [Online]. Available: https://doi.org/10.1016/j.datak.2021.101894

[10] G. Guizzardi, J. P. A. Almeida, N. Guarino, and V. Carvalho, "Towards an Ontological Analysis of Powertypes," in *Proc. Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology*, ser. CEUR Workshop Proceedings, vol. 1517, 2015.

[11] G. Guizzardi, "Ontological foundations for structural conceptual models," Ph.D. dissertation, University of Twente, 2015.

[12] B. Henderson-Sellers, *On the mathematics of modelling, metamodelling, ontologies and modelling languages*. Springer Science & Business Media, 2012.

[13] B. Kennel, "A unified framework for multi-level modeling," Master's thesis, Universität Mannheim, 2012. [Online]. Available: https://madoc.bib.uni-mannheim.de/31906/

[14] J. Odell, "Power Types," *Journal of OO Programming*, vol. 7, no. 2, pp. 8–12, 1994.

[15] OMG, *OMG Unified Modeling Language (OMG UML), Superstructure, Version 2.5.1*, Object Management Group Std., Rev. 2.5.1, December 2017. [Online]. Available: http://www.omg.org/spec/UML/2.5.1

[16] W3C, "OWL 2 Web Ontology Language - Document Overview (Second Edition)," 2012. [Online]. Available: http://www.w3.org/TR/owl2-overview/