Supporting Constructive and Exploratory Modes of Modeling in Multi-Level Ontologies

Colin Atkinson, Bastian Kennel, and Björn Goß

University Mannheim
A5,6, D-68131 Mannheim
{atkinson,bastian.kennel,bgoss}@informatik.uni-mannheim.de
http://swt.informatik.uni-mannheim.de

Abstract. In the long term, fully exploiting the potential synergies between ontology engineering and software engineering approaches to modeling is contingent on a complete understanding of their similarities and differences and the development of a unified modeling framework that can support both paradigms in a seamless way. In this paper we contribute to the former by characterizing the fundamental modes of modeling that the different approaches have evolved to support and we contribute to the latter by outlining how a promising unification paradigm can be enhanced to support them. This is the potency-based multi-level modeling paradigm which allows all ingredients of a comprehensive ontology (e.g. instances, classes and metaclasses) to be treated in a uniform way, and allows ontologies to include as many classification levels as needed to best represent the domain in hand. The paper explains how the basic notion of potency can be enhanced to support "exploratory" and "unbounded" modes of modeling alongside the "constructive" and "bounded" modes of modeling which underpin the software engineering approach to modeling for which it was originally developed.

Keywords: Multi-level modeling, metamodeling, modeling modes, constructive modeling, exploratory modeling, ontology engineering, potency

1 Introduction

As the potential synergy between ontology-based reasoning services (a la OWL) and software-engineering oriented specification capabilities (a la UML) becomes increasingly apparent, a growing number of researchers have started to explore ways of bringing the two schools of modeling together. In the short to medium term, "bridging" technologies that support mappings between the two paradigms offer the most practical way of leveraging these two forms of modeling together. Prime examples of such technologies include the Ontology Definition Metamodel standardized by the OMG [4] and the TwoUse interoperation tool [10] developed in the MOST project [6]. Nevertheless, in the long term, the cleanest solution is to develop a single unified modeling paradigm which incorporates the capabilities of existing modeling technologies as special cases.

Accommodating the different assumptions and interpretations underlying software engineering oriented modeling and ontology (i.e. semantic web) oriented modeling within a single unified framework presents several fundamental challenges, however. Some of these have been identified and widely discussed in the literature. Examples include different underlying logics to define the meaning of classes and taxonomies (e.g. description logic, basic set theory), different interpretation of missing information (e.g. closed-world versus open world assumptions) and different architectures used to organize model information (e.g. traditional four level OMG architecture), two level OWL architecture) [2, 14, 15].

Other differences, in contrast, have received relatively little attention and have yet to be fully elaborated. These relate to the fundamental modes by which models are developed and used in the software and ontology engineering communities and the criteria under which models are considered valid. One important difference relates to whether models are developed and used in a "constructive" or "exploratory" way. In constructive (i.e. software engineering oriented) modeling the role of class diagrams is to serve as templates from which populations of instances can be generated at a future point in time, while in exploratory (i.e. semantic web oriented) modeling the role of class models (i.e. ontologies) is to capture classification information wrapped up in an already existing population of instances. This is related to the distinction between "prescriptive" and "descriptive" models [18, 19], but focusses on the conditions under which model content can be considered complete or valid rather than on the purpose for which they are being deployed. Both prescriptive and descriptive models are often developed in exploratory mode and applied in constructive mode. Another important difference relates to whether the ways in which model content can be extended are "bounded" or "unbounded". Bounded models can only be extended across a predefined number of classification levels, while unbounded models can be extended over an unlimited number of classification levels.

In this paper we fully characterize these different modes of modeling and present a strategy for accommodating them seamlessly within a unified framework. This framework is based on the multi-level modeling paradigm that is emerging as the favored foundation for a unified modeling framework because it allows all ingredients of an ontology (e.g. instances, classes and metaclasses) to be treated uniformly as first class citizens, and allows an ontology to include as many classification levels as needed to best represent the domain in hand [13, 15]. The paper makes three concrete contributions – (1) it elaborates and fully characterize the different modeling modes, (2) it introduces and consolidates new terminology to discuss the properties of models in the context of these modes and (3) it enhances the notion of potency, the key features that support arbitrary numbers of classification levels, to support the unification of these modes within a single, integrated modeling framework.

2 Fundamental Modes of Modeling

As Figure 1 illustrates, there are two fundamental concerns that characterize the way models are developed and the assumptions under which they are used – the order or "direction" in which elements in the classification hierarchy are created and the "scope" of models in terms of their intended completeness and extensibility. These concerns are orthogonal and both give rise to a dichotomy of choices.

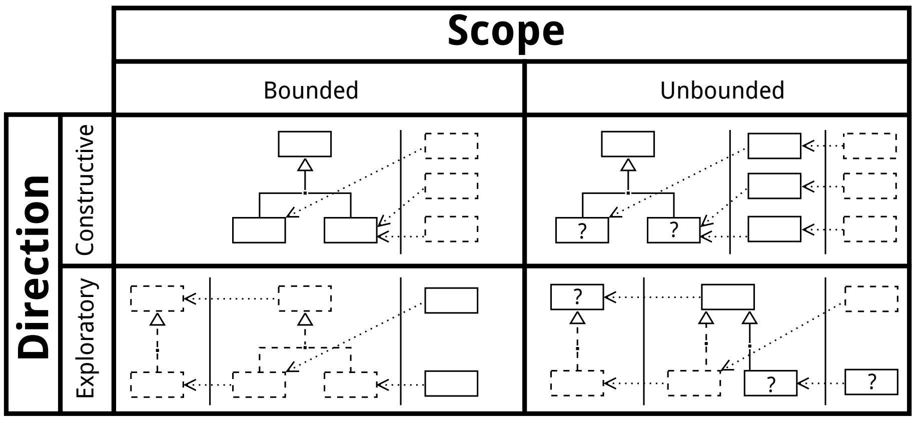

Fig. 1. Modeling Modes

2.1 Constructive versus Exploratory Modeling

The dominant dichotomy is the direction dichotomy which distinguishes between constructive modeling and exploratory modeling according to the general direction in which the elements in the classification levels are populated and the criteria under which a completed model may be considered valid.

Constructive Modeling The goal of constructive modeling is to create a complete, definitive description of all the types in a system (i.e. to create "type models" in the sense of Kühne [9]) so that instances can be generated from them at runtime. Since the instances are usually generated in a different runtime environment from that used to represent the types they are normally not regarded as being part of the model per se. For example, type models are frequently used to describe the objects that can populate a database or exist in the runtime system of a software application.

The idea behind constructive modeling is illustrated in the top left hand part of Figure 1 which shows two classification levels separated by a vertical line. Those on the left hand side of the line are types while those on the right hand side are instances. The solid and dashed boundaries of the elements are intended to show the chronological direction (order) in which the elements are created, with the solid elements being created before the dashed ones. Thus, this picture shows a type model from which a population of instances is subsequently created by instantiation.

An important convention in constructive modeling is that a type model is considered consistent and valid without any of the instances described by the model actually being present in the modeling framework. In other words, in constructive modeling the solid-lined elements at the type level in the upper left hand corner of Figure 1 can be considered complete and valid before any of the dashed instances at the instance level actually exist. Because all “instances” are created mechanically at runtime from types, another fundamental characteristic of constructive modeling is that every instance has a special relationship to one individual type. In constructive modeling a concrete class is a class that can be instantiated while an abstract one is a class that cannot be instantiated.

Since it evolved in the software engineering community, the UML is usually used in a constructive modeling mode and most of its features and terminology are optimized for this purpose. However, the UML can be, and often is, used in an exploratory way, for example during the analysis phase of a development project, although its semantics are not optimized for this mode of modeling.

Exploratory Modeling One aim of exploratory modeling is to develop types that characterize the objects populating a domain of interest. As illustrated in the bottom left hand corner of Figure 1, in contrast with constructive modeling where the types come before the instances, in exploratory modeling the types are usually identified from, and therefore after, the instances they classify. Moreover, a body of model content (i.e. an ontology) is only considered complete and valid when the instances classified by types are also present.

This distinction is not only reflected in the features of languages and tools used to create models, it is also reflected in the “mood” of natural language used to refer to the population of types and instances. Whereas in constructive modeling most of the terminology used to define language semantics and model properties is in the subjunctive form, in exploratory modeling it is in the assertive form. The derivation of types from instances is often performed by human ontology developers, but ontology engineering tools also typically provide “reasoning” services that can identify types and subtypes automatically.

The direction of population and causality in exploratory modeling means that the notion of mechanical instantiation is no longer relevant. Instead, instanceOf relationships need to be established based on the intention of types, applying the usual principle that an object is an instance of a type if it possesses all of the properties that the type defines for its instances. This means that instances no longer have a single, special type from which they were created. They are usually instances of multiple types, in different ways, depending on their properties. This also has an impact on the way subtyping and abstract/concrete classes are viewed. Because exploratory modeling focuses on describing typing concepts that exist in a domain, it is possible for two types to have the same intentions and extensions. In other words, two types can be equal in the terminology of

OWL. Also, since there is no notion of mechanical instantiation, the difference between abstract and concrete classes has to be defined in terms of the extent to which the instances of a type satisfy its intention.

2.2 Bounded versus Unbounded Modeling

The difference beteween bounded and unbounded modeling revolves around whether a model is developed under the assumption that the number of classification levels is fixed (at the time the model is developed) or not fixed. Of course, it is always possible to change a model, once completed, and add more classification levels if desired. For example, a finished, bounded model with two levels can always be turned into a new, bounded model with three levels. However, this is not what is intended here. The key difference is that unbounded models can be extended over an arbitrary number of classification levels without any of the existing model elements being changed, whereas bounded models cannot.

Bounded Modeling Bounded modeling takes place in the context of a fixed number of classification levels. This is illustrated in the two left hand diagrams of Figure 1. This is the main mode of modeling performed today, both in constructive and exploratory modeling and includes UML models in software engineering, ER models in data base design and OWL models (a.k.a. ontologies) in knowledge engineering. Moreover, most modeling projects are usually only concerned with two levels – a type level and an instance level. Since the advent of the UML, the use of more classification levels has become common in the context of language definition. The UML infrastructure has always nominally been based on four classification levels, while tools that focus on the flexible definition of domain specific languages usually work with three levels.

Unbounded Modeling Unbounded modeling takes place in the context of an unfixed or uncertain number of classification levels. In other words, an unbounded model is a completed model that explicitly indicates that the number of levels is unfixed or uncertain. This is indicated by the question marks appearing within the diagrams on the right hand side of Figure 1. The presence of the question marks is meant to symbolize the fact that one or more of the model levels have model elements that are explicitly indefinite about the number of classification levels in the subject of interest. In the case of constructive unbounded modeling (top right), this indefiniteness arises in terms of future instantiation of the types in a completed model. In other words, it exists when the model content is intended to represent a framework where future extension (by population of additional model classification level(s)) is anticipated but the exact number of levels is left undetermined. This kind of indeterminacy is common in libraries and frameworks, but the uncertainty is usually associated with specialization levels rather than instantiation. Standard programming languages such as Java, and language metamodels such as UML, possess a predefined set of types which are intended to be specialized over an indeterminate number of specialization levels. The notion of unbounded modeling is based on the same idea but with the indeterminacy also associated with the number of classification levels.

An important consequence of unbounded modeling is that the numbering of classification levels can no longer be fixed a priori. If the goal is to define model content to serve as an extensible framework, the predefined levels have to be numbered in such a way that any new levels can be added in a systematic way. A numbering scheme that fulfils this requirement is introduced at the end of Chapter 4.2.

3 Potency-based Multi-Level Modeling

Various flavors of multi-level modeling have been developed in recent years [11–13]. One of the most widely known is based on the notions of clabjects and potency to represent multiple classification levels [8, 4]. In recent years, a growing number of researchers have recognized this paradigm's potential to unify the ontology-engineering and software engineering approaches to modeling [1, 2, 15].

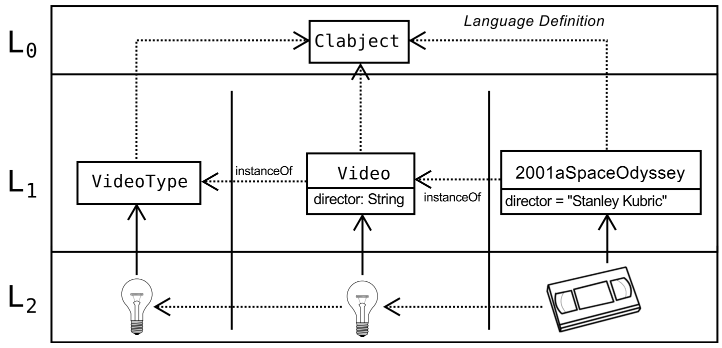

Fig. 2. Orthographic Classification Architecture

The key idea in this form of multi-level modeling is to recognize and separate two distinct forms of classification – one (linguistic classification) capturing the linguistic type of model elements, the other (ontological classification) capturing the ontological type. Since this leads to two orthogonal hierarchies of classification levels, each conforming to the tenets of strict metamodeling, the resulting architecture is sometimes referred to as the orthogonal classification architecture (OCA). This is depicted schematically in Figure 2, where the vertically arranged levels, L0, L1 and L2 represent the linguistic levels, and the horizontally arranged levels (within L1) represent the ontological levels. The bottom level, L2, represents the real world as in the UML model hierarchy. Clabjects are the basic modeling elements used to populate multi-level models in the OCA. They are able to do so in a level-agnostic way because in general, they represent classes and objects at the same time. All the main model elements in Figure 2, in all three ontological levels, are clabjects, and the element in the middle level, Video, has both a type and an instance facet since it is an instance of VideoType on the left hand side and a type for 2001aSpaceOdyssey on the right hand side. Clabjects can also have attributes (previously called fields [8]) and participate in connections.

3.1 Potency

Since clabjects are essentially level and type agnostic, some way is needed to represent the "typeness" of a clabject - that is, the degree to which it is capable of having instances. To achieve this Atkinson and Kühne introduced the notion of potency to indicate how many levels a clabject can have instances over [8]). This original notion of potency is based on the following concepts -

1. potency is a non-negative integer associated with clabjects and attributes,
2. a clabject, i, instantiated from another clabject, t, has a potency one less than t,
3. the potency of an attribute of a clabject must be no greater than that of the clabject,
4. every attribute of a clabject, i, instantiated from a clabject t, must have a corresponding attribute in t, with a potency one higher,
5. a clabject, i, instantiated from another clabject, t, must have an attribute corresponding to every attribute of t, except those that have potency 0.

"Corresponding" here means an attribute with the same name and datatype. Since potency values are defined as non-negative integers, it follows that a clabject of potency 0 cannot be instantiated (i.e. represents an object in classic UML terms) and a clabject of potency 1 represents a regular class in classic UML terms. By allowing clabjects to have potencies of 2 and higher, new kinds of classification behavior (spanning more than immediately adjacent levels) can be supported. This has been referred to as deep instantiation. This interpretation of potency also lends itself to a simple definition of abstract and concrete classes. Since abstract classes cannot be instantiated they have a potency of 0, while concrete classes which can be instantiated have a potency greater than 0. The basic problem with this original definition of potency is that it is inherently constructive and bounded. In other words it is inherently based on the principles that underpin the constructive and bounded modes of modeling. In fact, the very name "potency" conveys a constructive (type to instance) mentality. The fundamental definition of potency has always relied on the constructive notion of instantiation (instances created from types) and is not compatible with exploratory and unbounded modeling. In the following sections we therefore propose enhancements to the notion of potency to support all the modes of modeling identified above.

4 Constructive Multi-Level Modeling

To avoid ambiguity in the remainder of this paper we refer to a given ontological level as a model and the total collection of models, across all the ontological levels in the L1 level, as an ontology. This conforms to the prevailing notions of type and token models of Kühne [9] and the practice of considering instances as well as types when developing ontologies. The original definition of potency in the previous section is based on the notion that the extension of a type is characterized exclusively by attributes. However, in general this is an over simplification. If clabjects are allowed to have methods and enter into relationships with other clabjects these also have a bearing on the intention of a type. A full treatment of classification in the context of clabjects therefore needs to take these characteristics into account. However, these details are unimportant for understanding the modes of modeling discussed in this paper and the differences between them. Without loss of generality, therefore, in the remainder of the paper we assume that the only properties possessed by clabjects are attributes.

4.1 Bounded Constructive Modeling

The definition provided in the previous section does not really need to be changed to support constructive multi-level modeling it already contains the required ingredients. However, in order to define a unified framework and better capture bounded constructive modeling's place within it, it is helpful to introduce new terminology to distinguish the different interpretations of classification relationships employed in each mode. As described previously, classification in constructive modeling is used to drive the automated generation of instances in a running system. Since every clabject therefore has one-and-only-one type from which it is created, we refer to this type as the **blueprint** of the clabject. Going further, if one clabject, b, is a blueprint of another clabject, o, we say that o is an **offspring** of b.

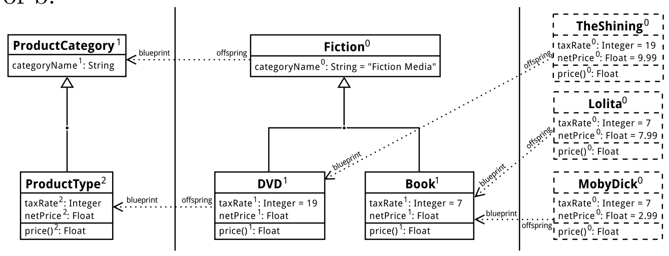

Fig. 3. Bounded Constructive Modeling

Figure 3 shows an example of a bounded, multi-level ontology that has been created in constructive mode. This is represented in the LML (Level-agnostic Modeling Language) that we have optimized to visualize multi-level ontologies [16]. The only aspect of this diagram that is not valid LML syntax is the representation of clabjects with dashed boundaries on the right hand side of the model. This notation is used here to convey the idea that the instances were instantiated from, and thus created after, the types in the middle. As always some exceptions are required at the extreme end of the model hierarchy and the clabjects in the left most model do not have blueprints. In terms of these new definitions we can interpret the potency of a clabject as defining the maximum potential depth of its deep offspring tree – that is the tree formed by taking all its offsprings, and all of its offspings' offsprings, and so on, recursively.

4.2 Unbounded Constructive Modeling

If it were ever deemed necessary to add a fourth model level to the bounded ontology in Figure 3 it would be necessary to change the potency values of many of the existing clabjects, leading to new versions of the type models in the two left hand levels. However, a common goal in constructive software engineering is to create reusable frameworks containing models that can be extended in arbitrary ways without requiring them to be changed. The extension is usually achieved by specialization, for example in the definition of Java class libraries or in the addition of new abstract syntax elements to language definitions. Extension by specialization is inherently unlimited and does not require changes to existing model elements because they do not carry any predetermined constraints on the depth of the specialization hierarchy. However, with the bounded form of potency described previously, it is not possible to add more ontological level to an ontology without changing the potencies of the existing model elements, defeating the goal of defining predefined frameworks of model content which can subsequently be extended as necessary in future projects.

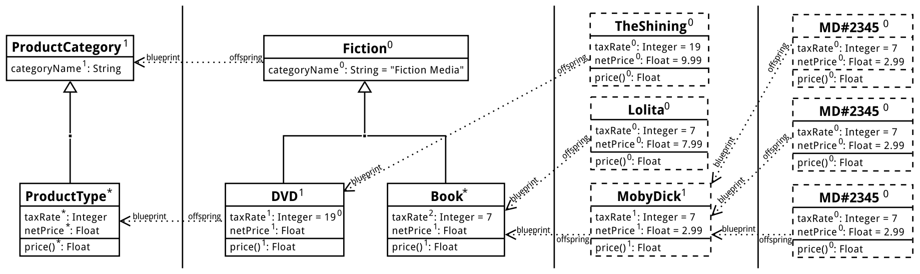

Fig. 4. Unbounded Constructive Modeling

To address this problem we extend the potency concept defined previously to include the notion of unlimited or * potencies. These have the same basic meaning as * multiplicity constraints in UML. Instead of leaving the number of instances of an association open, however, * multiplicities leave the number of ontological levels open. The use of * multiplicities is illustrated in Figure 4 above. The stars can be thought of as the concrete incarnation of the question marks in the schematic representation of the modeling approaches in Figure 1. Figure 4 is quite similar to Figure 3, but obviously has four ontological levels. This is possible because the potencies of ProductType and Book are * rather than 2 and 1 respectively. Of course, it would be possible to create a well formed model by changing the potencies of ProductType and Book to 3 and 2, but this would give rise to another bounded model which could not be extended in the future. By making the potencies of these clabjects * an unlimited number of levels can be added without change. In Figure 4, the two left hand levels can be interpreted as defining an extensible framework. This framework can support the right hand model of Figure 3 as well as the two right hand models of Figure 4. To accommodate unbounded multi-level modeling using * potencies, the overall definition of potency needs to be enhanced as follows -

1. Potency is a non-negative Integer or a * associated with clabjects and attributes,
2. An offspring of a clabject with numeric potency must also have a numeric potency with a value that is one lower,
3. An offspring of a clabject with * potency may also have * potency or any numeric potency,
4. the potency of an attribute of a clabject must be no greater than that of the clabject,
5. a clabject, i, instantiated from another clabject, t, must have an attribute corresponding to every attribute of t, except those that have potency 0. If the attribute of t has * potency the corresponding attribute of i can have * potency or any numeric potency. If the attribute of t has a (non-zero) numeric potency, the corresponding attribute of i must have a numeric potency that is one lower.

Extending the potency mechanism in this way with the notion of * potencies has the desired effect of allowing predefined models to be extended in unpredicted ways, but it also raises a dilemma related to the numbering of the ontological levels. Since they all implicitly assumed a bounded mode of multi-level modeling, existing modeling infrastructures such as that of the UML and the original OCA chose a fixed numbering scheme in which the most general levels have the highest numbers and the least abstract levels instantiated from them have the lowest numbers, with the bottom level having the number 0. However, this is obviously not extensible across an arbitrary number of levels, unless one accepts levels having negative numbers which is somewhat counter intuitive. To support unlimited extension of generic models across an unlimited number of levels it is therefore necessary to adopt a number scheme in which the less abstract (instantiated) levels have a higher number. In terms of Figure 4, this would mean that the left hand model would have level number 0, the model to the right, instantiated from it, would have level number 1 and so on. In this way, the number of levels could grow in an unlimited way. This reverses the numbering convention popularized by the UML infrastructure.

5 Exploratory Multi-Level Modeling

The notion of mechanically instantiating types to generate instances is not relevant in exploratory modeling. Instead, a more discovery-oriented notion of classification (instanceOf) is required based on the properties that a clabject requires to be considered an instance (i.e the extension of a type). The basic requirement that an instance of a type needs to posses all the properties defined in the extension of that type remains unchanged. However, this notion alone is not sufficient to define useful semantics for potency in an exploratory context. It is also necessary to distinguish between two forms of instanceOf relationship one in which an instance has exactly the properties needed to be an instance of the type, and no more, and one in which an instance has more properties than those needed to be an instance of a type. To this end we introduce two distinct forms of classification (instanceOf) relationship - if a clabject has exactly the required properties needed to be an instanceOf a type but no more, we refer to it as an **isonymic instance** of that type, whereas if a clabject has more properties than those defined in the extension of a type, we refer to it as a **hyponymic instance** of that type. The set of instances of a type is thus partitioned into two subsets the set of isonymic instances and the set of hyponymic instances (i.e. isonyms and hyponyms for short).

5.1 Bounded Exploratory Modeling

These specialized forms of instanceOf relationship allow exploratory multi-level modeling to be supported in a simple and powerful way. Basically, instead of defining the semantics of potency in terms of the instantiatability of clabjects, in exploratory modeling it is characterized in terms of the depth of the isonymic instance tree emanating from a clabject.

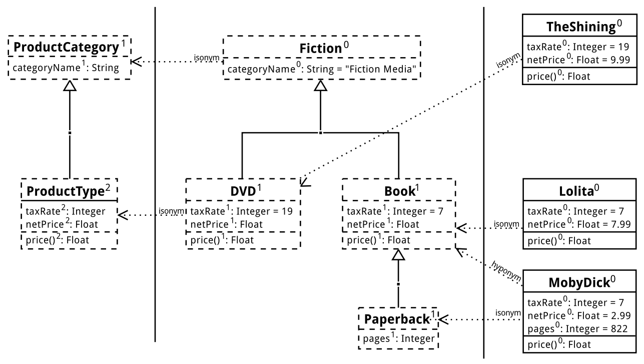

Fig. 5. Bounded Exploratory Modeling

The detailed definition of potency for bounded exploratory modeling is as follows -

1. potency is a non-negative integer associated with clabjects and attributes,
2. an isonymic instance of a clabject must have a potency one lower than the clabject,
3. the potency of an attribute of a clabject must be no greater than that of the clabject,
4. every attribute of a clabject, i, that is an isonymic instance of a clabject t, must have a corresponding attribute in t, with a potency one higher,
5. a clabject, i, that is an instance of a clabject, t, must have an attribute corresponding to every attribute of t, except those that have potency 0.

In contrast with the definitions in section 4.1, these definitions are driven by the isonymic instanceOf relationship rather than the instantiation relationship. The main consequence of this difference is that in exploratory modeling a clabject can have more than one isonymic type (which by definition are then equal types in the sense of OWL). Also, as an abstract type is defined as a type that has no isonymic instances (but possibly many hyponymic instances), while a concrete class is a type that has at least one isonymic instance. Because potency is defined in terms of isonymic instances, and as before always goes down by one across classification levels, it follows that it is a measure of the depth of the isonymic instance tree of a clabject. These concepts are illustrated in Figure 5 which actually contains the same clabjects as Figure 3. The difference is the direction in which the clabjects are interpreted as having been discovered and the ways in which the classification relationships are interpreted. In Figure 5 the clabjects in the two left hand models are dashed, indicating that they were derived from their instances in the model on the right hand side. Furthermore, the instanceOf relationships in the ontology are classified as being either isonymic or hyponymic. Notice that the extension of a clabject (i.e. the properties that the clabject requires of its instances) is defined by its attributes of potency 1 or higher. The potency zero attributes of a clabject have no effect on the instanceOf relationships it can participate in as a type.

5.2 Unbounded Exploratory Modeling

By analogy with unbounded constructive modeling in section 4, unbounded exploratory modeling extends the bounded form with the notion of * potency. However, whereas all the other three modes of modeling have a clear and well established role in information modeling and are all used in practical IT projects, the role of unbounded, exploratory modeling is less clear. We have included it in the scheme for symmetry and completeness, and because it fits within a rounded and clean extension of potency. The value of this mode of modeling is yet to be explored. As indicated in Figure 6, the main difference to unbounded constructive modeling is that the ontology is not populated in a specific direction (or order).

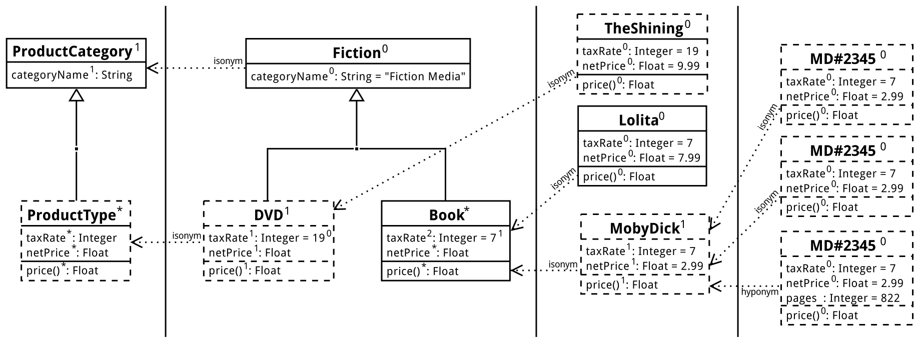

Fig. 6. Unbounded Exploratory Modeling

The detailed definition of potency for unbounded exploratory modeling is adapted from that for constructive modeling by the same principle of using the isonymic instanceOf relationship rather than instantiation -

1. potency is a non-negative integer or a * associated with clabjects and attributes,
2. an isonymic instance of a clabject with numeric potency must have a numeric potency with a value that is one lower,
3. an isonymic instance of a clabject with * potency may also have * potency or any numeric potency,
4. the potency of an attribute of a clabject must be no greater than that of the clabject,
5. every attribute of a clabject, i, that is an isonymic instance of a clabject t, must have a corresponding attribute in t, with a potency one higher. If the attribute of i has * potency the corresponding attribute of t can have * potency or any numeric potency.
6. a clabject, i, that is an isonymic instance of a clabject, t, must have an attribute corresponding to every attribute of t, except those with potency 0.

The interpretation of the * potency is different. Whereas in constructive modeling, * potency reflects a capacity to spawn offspring in an unbounded number of derived levels, in exploratory modeling it reflects a measure of uncertainty in how many levels there actually are in the subject of interest. This is related to the idea of open world interpretations of ontologies in knowledge engineering. However, the difference between open world and closed world interpretations of models revolves around how a modeling systems responds to queries in the absence of explicit information, while the uncertainty in exploratory, unbounded modeling revolves around the depth of isonym trees in multi-level ontologies.

6 Multi-Mode Modeling

At first sight the different interpretations and definitions of potency presented in the previous two sections might seem incompatible. With the enhancements suggested in this paper they can be integrated within a single framework, in which constructive modeling is essentially a special case of exploratory modeling. For example, the blueprint/offspring notion in the constructive view of potency is a special case of the isonymic/hyponymic instance dichotomy since by definition an offspring is an isonym. Having the basis for a framework in which all four modeling modes can be applied to the same content does not necessarily make it a good idea. It begs the question: why would a unified framework supporting all four modes of modeling be of value? We believe there are many scenarios where such capabilities are of value, but for space reasons describe only two of them below.

In most full spectrum software and database engineering projects the early phase of development starts out with an analysis or requirements elicitation phase where the basic properties of the problem to be solved and the systems requirements need to be understood. In other words, most projects start out in an exploratory mode where the goal is to learn and characterize the types in the domain of interest rather than define instantiatable types. But later on, once the project moves into the design phase, the goal is to move into a constructive mode and actually define the types that will be instantiated in the final database or software system. In software engineering this distinction is often characterized in terms of the difference between the domain model and the design class diagrams [17]. Thus, there is an inherent and natural need to switch between exploratory and constructive modes of development in most software and database engineering projects. Ideally, engineers would like to be able to populate an ontology in an exploratory way near the beginning of a project and then at some point move to a constructive approach.

Another important trend in the knowledge engineering and model-driven development communities is to explore how model content can be used at run-time, with associated reasoning and checking facilities, to drive the execution of software systems. This research goes under the name of "models@runtime" or "ontology-driven systems", and the resulting applications are often referred to as "intelligent" because they are able to exhibit behavior that is not hard-wired into them. Once again, this not only requires the ability to switch between constructive and exploratory modes of modeling (since the run-time ontology probably needs to be manipulated under the mechanisms for exploratory modeling) it also requires unbounded exploratory modeling since the direction and extent of an ontologys growth is unknown a priori.

7 Conclusion

In this paper we have identified four fundamentally distinct modes of modeling, based on the interaction of two orthogonal dichotomies that reflect the scope of models and the direction in which they are populated. We also presented examples of scenarios in which these modes are needed and where the ability to switch seamlessly between them would greatly increase the efficiency of software and data engineering projects. We then presented a generalization of the potency-based approach to multi-level modeling that equips it with the ability to accommodate the different modes within a single, unified, multi-mode modeling framework. This unified framework opens up the possibility of a truly fundamental unification of the ontology engineering and software/database engineering approaches to modeling in which the advantages of one community's technology can be exploited by the other and vice versa. In particular, the exploratory definition of potency based on the distinction between isonymic and hyponymic instanceOf relationships provides the basis for reasoning services of the kind available with OWL DL. The described approach therefore for the first time makes potency-based multi-level modeling useful to the knowledge engineering community as well as to the software and data engineering communities. We are currently working on an EMF-based prototype of the infrastructure which should make the approach usable within the rich set of facilities provide by the Eclipse modeling tools. We are also working on identifying stronger uses case for the unbounded exploratory mode of modeling and on understanding how these modes relate to the open and closed querying techniques applied to models.

References

1. Saeki, M. and Kaiya, H.: On Relationships among Models, Meta Models and Ontologies. In: Proceedings of DSM'06@OOPSLA2006 - 6th OOPSLA Workshop on Domain-Specific Modeling, Portland, Oregon USA (2006)
2. Atkinson, C., Gutheil, M., and Kiko, K.: On the Relationship of Ontologies and Models. 2nd International Workshop on Meta-modeling and Ontologies (WoMM2006), Karlsruhe (2006)
3. Weishan Z.; Hansen, K.M.; Fernandes, J.: Towards OpenWorld Software Architectures with Semantic Architectural Styles, Components and Connectors. In : ICECCS '09 Proceedings of the 2009 14th IEEE International Conference on Engineering of Complex Computer Systems, vol ., pp.40-49. IEEE Press, New York (2009)
4. Object Management Group, Inc.: Ontology Definition Metamodel. (2009)
5. Dong, J.S., Lee, C. H., Lee, H. B., Li, Y. F., Wang, H.: A combined approach to checking web ontologies. In: Proceedings of the 13th international conference on World Wide Web, New York, NY, USA (2004)
6. MOST Project - Marrying Ontology and Software Technology, http://www.most-project.eu
7. World Wide Web Consortium (W3C), http://www.w3.org/TR/rdf-schema/
8. Atkinson, C., Kühne, T.: Model-Driven Development: A Metamodeling Foundation. In: IEEE Software, 20(5), 3641 (2003)
9. Kühne, T.: Matters of (Meta-)Modeling. In: Journal on Software and Systems Modeling, Volume 5, Number 4, 369385 (2006)
10. Parreiras, F., Fernando, Staab, S., Winter, A.: TwoUse: Integrating UML Models and OWL Ontologies. Institut für Informatik, Universität Koblenz-Landau. Nr. 16/2007. Arbeitsberichte aus dem Fachbereich Informatik (2007)
11. Neumayr, B., Grün, K., Schrefl, M.: Multi-level domain modeling with m-objects and m-relationships. In: Proceedings of the Sixth Asia-Pacific Conference on Conceptual Modeling, Wellington, New Zealand (2009)
12. de Lara, J., Guerra, E.: Deep meta-modeling with METADEPTH. In: Vittek, J. (ed) TOOLS'10 Proceedings of the 48th international conference on Objects, models, components, patterns. LNCS, pp. 1-20. Springer, Heidelberg (2010)
13. Atkinson, C, Gutheil, M, Kennel, B.: A Flexible Infrastructure for Multi-Level Language Engineering. In: IEEE Transactions on Software Engineering, vol. 35, no. 6, pp. 742-755 (2009)
14. Staab, S., Walter, T., Gröner, G.,Parreiras, F.. Model driven engineering with ontology technologies. In: Reasoning Web. Semantic Technologies for Software Engineering, volume 6325 of LNCS. Springer (2010)
15. Jekjantuk, N., Gröner, G., Pan, J. Z., Zhao, Y.. Modelling and validating multilevel models with owl fa. In: Proceedings of the 6th International Workshop on Semantic Web Enabled Software Engineering (2010)
16. Atkinson, C., Kennel, B., Goß, B.: The Level-agnostic Modeling Language. In: Proceedings of the SLE 2010. LNCS, vol. 6563, Springer, Heidelberg (2010)
17. Larman, C.: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process. Prentice Hall PTR Upper Saddle River, NJ, USA (2001)
18. Seidewitz, E.: What models mean. In: IEEE Software 20(5), 26-32 (2003)
19. Assmann, U., Zschaler, S., & Wagner, G. (2006). Ontologies, Meta-models, and the Model-Driven Paradigm. Ontologies for Software Engineering and Software Technology, 249-273