Multi-Level Domain Modeling with M-Objects and M-Relationships

Bernd Neumayr **Katharina Grün** **Michael Schrefl**

Department of Business Informatics - Data & Knowledge Engineering,
Johannes Kepler University Linz, Austria
E-Mail: {neumayr,gruen,schrefl}@dke.uni-linz.ac.at

Abstract

Using traditional semantic data modeling, multi-level modeling can be achieved by representing objects in different abstraction hierarchies, namely classification, aggregation and generalization. This, however, leads to accidental complexity, complicating maintenance and extension. Several modeling techniques, like deep instantiation, powertypes and materialization, have been proposed to reduce unnecessary complexity in modeling objects at multiple levels. Multi-level objects (m-objects) and multi-level relationships (m-relationships) build on these results and provide a natural, intuitive representation of the *concretization* of objects and relationships along multiple levels of abstraction. By integrating aspects of the different abstraction hierarchies in a single concretization hierarchy, they improve readability and simplify maintenance and extension as compared to previous approaches. The discussion on conceptual modeling is complemented by a brief presentation of M-SQL, a data manipulation and query language for working with m-objects and m-relationships in an object-relational setting.

Keywords: Conceptual Modeling, Multi-Level Modeling, Multiple Abstraction

1 Introduction

Modeling domain objects at multiple levels of abstraction has received increased attention over the last years. It is nowadays also referred to as *ontological* multi-level modeling to contrast it from *linguistic* meta-level modeling, which relates to representing modeling language constructs in one or more higher levels, or meta models. Note that multi-level modeling is often associated solely with multiple classification levels, while this paper concerns, more generally, levels in classification, generalization and aggregation hierarchies.

Objects are frequently organized in hierarchies consisting of multiple levels. Examples are product hierarchies, dimension hierarchies in data warehouses, and taxonomies in general. E.g., a *product catalog* may consist of three levels, *category*, *model*, and *physical entity*. Sample instances of these levels could be *car*, *porsche911CarreraS*, and *myPorsche911CarreraS*, respectively.

Conceptual modeling of such hierarchies is straightforward if objects at each level are *uniform*, i.e. have the same structure. As information systems grow in size or previously independent systems are integrated across related domains, the need to handle levels of similar, yet non-uniform objects arises. Objects at the same level in different subhierarchies may differ from each other. E.g., *product model*s have a *listprice*, but *car model*s (i.e., objects at level *model* belonging to *car*, an object at level *category*) have an additional attribute *maxSpeed*.

Non-uniformity may go beyond having differently described objects at the same level. Sub-hierarchies beneath two objects at the same level may differ from each other in that they introduce different additional levels. E.g., our product catalog may contain a product category *car*, which is described by an additional level *brand* (with objects like *porsche911*) between levels *model* and *physical entity*.

Such non-uniform hierarchies of domain objects at multiple levels of abstraction can be modeled using general modeling languages like UML through a combination of aggregation, generalization and classification (see left side of Fig. 1 for a simplified UML representation in which additional level *brand* is omitted). While, from a static perspective, this UML representation does not involve unnecessary or *accidental complexity*, the accidental complexity becomes apparent when introducing new domain concepts. Introducing a single domain concept (such as new product category *car*) requires next to introducing an instance *Car:CarCategory* to add multiple classes, (in our case *CarCategory*, *CarModel*, *CarPhysicalEntity*) as well as associated aggregation, generalisation, and instantiation relationships. Moreover, the same kind of classes need to be added for each new product category entered into our product catalogue. The redundancy and fragmentation caused by following such a modeling approach complicates maintenance and extension.

Several modeling techniques have been proposed in recent years to reduce accidental complexity in modeling domain objects at multiple levels of abstraction. The prevailing techniques are *materialization* (Goldstein & Storey 1994, Pirotte et al. 1994, Dahchour et al. 2002), *powertypes* (Odell 1998, Henderson-Sellers & Gonzalez-Perez 2005, Gonzalez-Perez & Henderson-Sellers 2006), and *deep instantiation* (Atkinson & Kühne 2001, Kühne & Schreiber 2007). While these techniques simplify multi-level modeling and support deep characterisation, they do not support (or at least do not directly support) non-uniform sub-hierarchies. *Materialization* provides for modeling concrete objects of category objects differently and comes with a very powerful concept for property definitions and propagations along the materialization hierarchy. The *powertype* approach provides for describing the common properties of subclasses of a class, i.e., one level of the generalization hierarchy, by powertypes. Ontological meta modeling

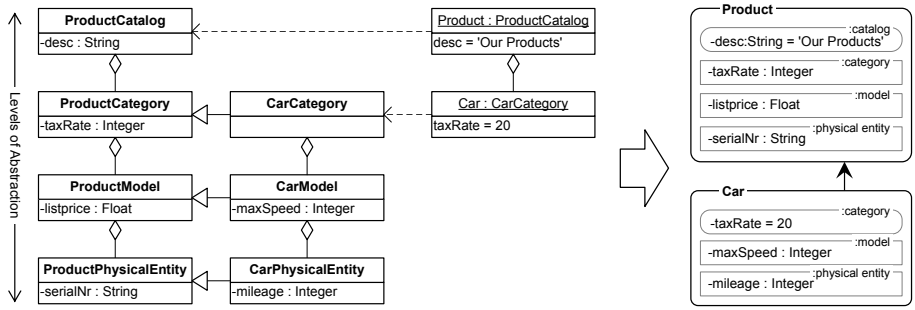

Figure 1: Basic Idea of Multi-Level Objects (simplified), Left: UML, Right: M-Objects

with *deep instantiation* provides for multiple levels of classification whereby an object at one level can describe the common properties for objects at each instantiation-level beneath that level.

Multi-level objects (m-objects) and *multi-level relationships* (m-relationships), introduced in this paper, build on these approaches and provide a natural, intuitive presentation of the *concretization* of objects and relationships along multiple levels of abstraction. The basic ideas of our approach are (i) to encapsulate the different levels of abstractions that relate to a single domain concept (e.g., the level descriptions, *catalog*, *category*, *model*, *physicalEntity* that relate to single domain concept, product catalog) into a single m-object (cf. *Product* with these levels at the right hand side in Fig. 1), and (ii) to integrate aspects of the different semantic abstraction hierarchies (aggregation, generalization, and classification) in a single *concretization* hierarchy. Note that we use the term aggregation in a broad sense referring to abstraction levels both with a typical whole-part flavor (e.g. *city-county-state*) or with a materialization flavor (e.g. *physical entity-model-category*). Furthermore our approach provides a naming scheme to address and query meta^n-classes which are implicitly introduced with the levels of abstractions of a m-object.

A concretization relationship between two m-objects does not reflect that one m-object is at the same time an *instance of*, *component of*, and *subclass of* another m-object as a whole. Rather, a concretization relationship between two m-objects, such as *Car* and *Product* in Fig. 1, is to be interpreted in a multi-faceted way. M-object *Car* is an instance of m-object *Product* with respect to that m-object's second-top level, *category*, and gives values for attributes of that level, e.g., *taxRate*. M-object *Car* is a component of m-object *Product* considered as an object of that m-object's top-level, *catalog*. Each level of m-object *Car* can be seen as a subclass of the corresponding level of m-object *Product*.

Relationships between m-objects are likewise described at multiple levels, associating one or more pairs of levels of the linked m-objects, and are, thus, called m-relationships. M-relationships can again be interpreted in a multi-faceted way, once as relationship occurrence (or sometimes also called relationship instance or link) and once as relationship class (or sometimes also called relationship type, or set), or also as meta-relationship class. They take on these multiple roles depending on what pairs of linked levels one considers and on whether one considers the linked m-objects in an instance or a class-role. M-relationships may be concretized along the concretization hierarchy of linked m-objects. The richness of this concise, powerful modeling concept is discussed in detail in the paper.

The paper is organized as follows: Section 2 defines requirements for multi-level modeling based on a sample problem. Section 3 presents the concept of m-objects and their consistent concretization in concretization hierarchies. Section 4 introduces m-relationships. Section 5 outlines a data definition and manipulation language M-SQL for m-objects. Section 6 gives an overview of related work and compares it to our approach. Finally, Section 7 concludes the paper.

2 Sample Problem and Requirements

In this section, we present our running example and discuss requirements for multi-level modeling approaches.

Example 1 (Sample Problem)**.** A sample online-store buys and sells *products*, which are described on at least three levels of abstraction: *physical entity*, *model*, and *category*. Each *product category* has associated a *tax rate*, each *product model* has a *list price*. Book editions, i.e. objects at level *model* that belong to *product category book*, additionally have an *author*. Our company keeps track of *physical entities* of *products* (e.g. copies of book *Harry-Potter4*), which are identified by their *serial number*. In addition to *books*, our company starts to sell *cars*, i.e. it introduces *car* as a new *product category*. Cars differ from books in that they are described at an additional level, namely *brand*, and in that they have additional attributes: *maxSpeed* at level *product model* and *mileage* at level *physical entity*. As our sample-online-store specializes on selling cars of *brand Porsche 911*, it wants to be able to register physical entities of this *car brand* at the *Porsche 911 club*. Our company further keeps track of *companies* that produce these products. Companies are likewise described at multiple levels: *industrial sector*, *enterprise*, and *factory*. Producers of cars belong to a specific *industrial sector*, namely *car manufacturer*. To track quality problems, our company also associates with each *physical entity* of *category car* the *factory* at which it was produced. Thereby, this *factory* must belong to the *enterprise*, which produces the *car model*.

To model domains, such as described in Example 1, multi-level modeling approaches may be benchmarked against how they support the following requirements:

- *Multiple levels of abstraction:* It should be possible to describe objects at different levels of abstraction. For example, products are described at levels *category*, *model*, and *physical entity*.
- *Extensibility:* Multi-level modeling should offer extensibility to model non-uniform sub-hierarchies, i.e. to add levels, attributes or relationships. For example, cars are described by an additional level, namely *brand*, and have additional attributes, like *maxSpeed*.

- *Avoid fragmentation and redundancy:* Every information concerning one domain object should be described local to that object to avoid fragmentation and redundancy. E.g., when introducing product category car, its information should be described in one object, which avoids distributing information along various objects and redundant modeling of relationships (cf. Figure 1).
- *Relationships:* The modeling approach should support relationships between objects and their specialization at different levels of abstraction. For example, it should be possible to associate products with companies and to further specialize this relationship by specifying that cars can only be produced by car manufacturers.
- *Queries:* To work with multi-level models, support for queries and navigation between levels is required. Queries can refer to different levels and objects, e.g. to retrieve all car models (i.e. all objects at level *product model* belonging to *product category car*). To determine e.g. the *taxRate* applying to *myPorsche911CarreraS*, navigation between levels of abstraction is necessary.

3 Multi-Level Objects

Based on multi-level modeling requirements presented in Section 2, we introduce, exemplify and formally define m-objects and their concretization in this section. We first describe m-objects and how one m-object can concretize another m-object. We then look at the roles which a m-object plays in such a concretization. Building on a formal definition of m-objects we specify rules for consistent concretization of m-objects and for consistent concretization hierarchies. Note that the basic concepts of m-object hierarchies could alternatively be described by providing a mapping to UML and OCL, or to the Higher-order Entity-Relationship Model (HERM) (Thalheim 2000). Our concise definitions are independent of a specific conceptual modeling language and well suited for defining consistency rules and operators for working with m-objects (cf. Section 5).

A m-object encapsulates and arranges abstraction levels in a linear order from the most abstract to the most concrete one. Thereby, it describes itself and the common properties of the objects at each level of the concretization hierarchy beneath itself. A m-object that concretizes another m-object, the parent, inherits all levels except for the top-level of the parent. It may also specialize the inherited levels or even introduce new levels. A m-object specifies concrete values for the properties of the top-level. This top-level has a special role in that it describes the m-object itself. All other levels describe common properties of m-objects beneath itself.

Definition 1 (M-Object)**.** *A m-object* $o = (L_o, A_o, p_o, l_o, d_o, v_o)$ *consists of a set of levels* L_o*, taken from a universe of levels* L *and a set of attributes* A_o*, taken from a universe of attributes* A*. The levels* L_o *are organized in a linear order, as defined by partial function parent* $p_o : L_o \rightarrow L_o$*, which associates with each level its parent level. Each attribute is associated with one level, defined by function* $l_o : A_o \rightarrow L_o$*, and has a domain, defined by function* $d_o : A_o \rightarrow D$ *(where* D *is a universe of data types). Optionally, an attribute has a value from its domain, defined by partial function* $v_o : A_o \rightarrow V$*, where* V *is a universe of data values, and* $v_o(a) \in d_o(a)$ *iff* $v_o(a)$ *is defined.*

We alternatively represent the order of levels by parent relation $P_o \subseteq (L_o \times L_o)$, where $(l', l) \in P_o$ iff $p_o(l') = l$; and denote its transitive closure by P_o^+ and its transitive and reflexive closure by P_o^*. We denote the top-level by $\hat{l}_o$ and the set of attributes associated with the top-level by $\hat{A}_o$. Further, we say that a m-object is at level l if l is its top-level. We denote by O the set of m-objects. M-objects, levels, and attributes have names, defined by function $n : O \cup L \cup A \rightarrow N$, where N is the universe of names. Levels of the same m-object must have distinct names. The names of attributes that are associated with the same level of a m-object must be unique as well.

Example 2. Consider m-object *Product* of Figure 2, which consists of four levels with one attribute each. Level *category* is the parent level of level *model*. Attribute *taxRate* is associated with level *category*, has domain *Integer* and does not define a value. The m-object is at level *catalog* as this level is its top-level. The top-level defines attribute *desc* of domain *String* and specifies value *'Our Products'* for this attribute.

A m-object can *concretize* another m-object, which is referred to as its parent. A concretize-relationship comprises classification-, generalization- and aggregation-relationships between the levels of a m-object and the levels of its parent, as follows:

- *Classification - Instantiation:* Each m-object can be regarded as an instance of its parent m-object. In particular, the top-level of a m-object is an instance of the second top-level of its parent m-object. A m-object must adopt all levels from its parent except for the parents top-level and it can specify values for its attributes.
- *Generalization - Specialization:* The level descriptions of a m-object correspond to subclasses of the corresponding levels of its parent. The m-object can define new levels or add attributes to levels. Thereby, a m-object must not change the relative order of levels it inherits from its parent.
- *Aggregation - Decomposition:* The concretization path between m-objects of different levels expresses an aggregation hierarchy at the instance level. At the schema level the aggregation hierarchy is given by the order of levels of a m-object.

Example 3 (concretization)**.** M-object *Car* concretizes m-object *Product* in Figure 2. The concretization relationship expresses classification: m-object *Car* is instance of level *category* of m-object *Product* because level *category*, which is the first non-top-level of m-object *Product*, is its top-level. It also specifies a value for its attribute *taxRate*. M-object *Car* specializes m-object *Product* by introducing a new level *brand* and adding attribute *maxSpeed* to level *model* and attribute *mileage* to level *physical entity*. The level *model* of m-object *Car* can be regarded as a subclass of level *model* of m-object *Product*. Cars are further categorized into brands, models and physical entities. These levels define the aggregation hierarchy.

When organizing m-objects in a hierarchy, the name of each m-object, given by function $n : O \rightarrow N$, must be unique within the direct descendants of its parent. M-objects organized in a concretization hierarchy inherit properties and functions from their parent m-objects and, thus, may only be partially defined. A child m-object o' inherits from its parent m-object o all properties, i.e. levels, attributes (and relationships, cf. Section 4), and function definitions

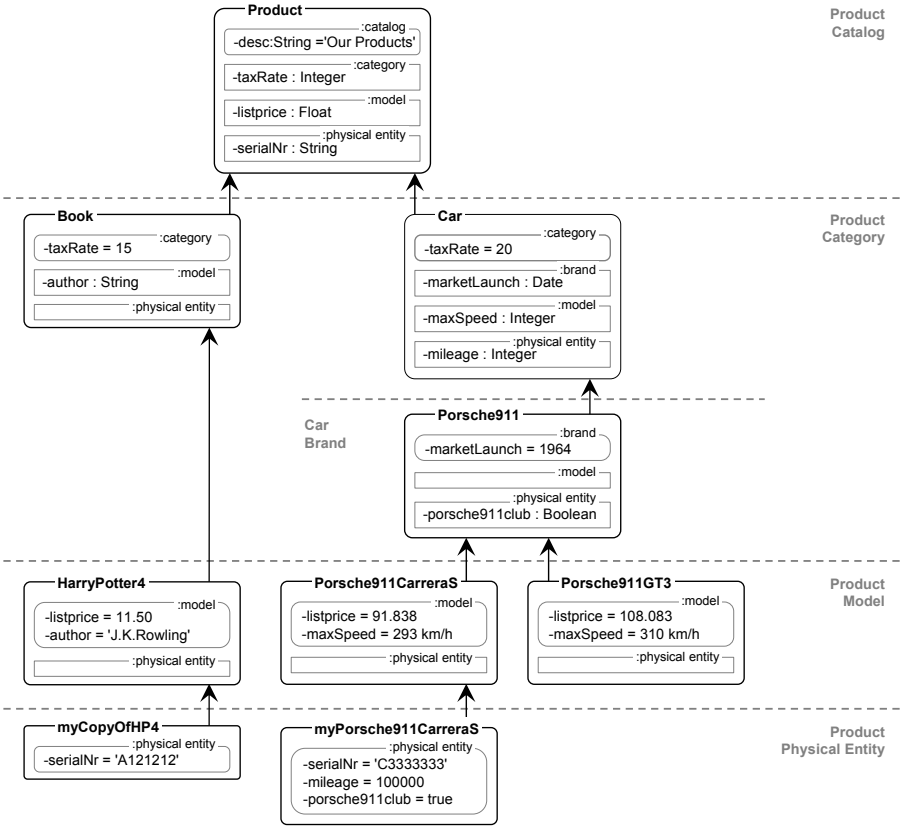

Figure 2: Concretization hierarchy of a product catalog along multiple levels (inherited attributes not shown)

beyond the parent's top-level. The inherited properties are available in all function definitions of the child ($p_{o'}$, $l_{o'}$, $d_{o'}$, $v_{o'}$). These functions, if only partially specified at the child, are extended for undefined arguments using the corresponding functions of the parent (p_o, l_o, d_o, v_o). In the following definition of a consistent concretization, we assume that partially defined m-objects have already been extended as just described.

Definition 2 (Consistent Concretization)**.** *A m-object o' is a* consistent concretization *of another m-object o iff*

1. *Each level of o, except for the top-level, is also a level of o':* $(L_o \setminus \{\hat{l}_o\}) \subseteq L_{o'}$ *(*level containment*)*
2. *All attributes of o, except for the attributes of the top-level, also exist in o',* $(A_o \setminus \hat{A}_o) \subseteq A_{o'}$ *(*attribute containment*)*
3. *The relative order of common levels of o and o' is the same:* $l, l' \in (L_{o'} \cap L_o) : (l, l') \in P^+_{o'} \rightarrow (l, l') \in P^+_o$ *(*level order compatibility*)*
4. *Common attributes are associated with the same level, have the same domain, and the same value, if defined: For* $a \in (A_{o'} \cap A_o)$*:*
   (a) $l_o(a) = l_{o'}(a)$ *(*stability of attribute levels*)*
   (b) $d_o(a) = d_{o'}(a)$ *(*stability of attribute domains*)*
   (c) $v_o(a)$ *is defined* $\rightarrow v_o(a) = v_{o'}(a)$ *(*compatibility of attribute values*)*

Example 4 (consistent concretization)**.** M-object *Car* is a consistent concretization of m-object *Product* in Figure 2. Except for the top-level *catalog*, each level of *Product* also exists in *Car* (level containment). The same is true for all attributes, i.e. *Car* only misses attribute *desc*, which is a top-level attribute of *Product* (attribute containment). The levels have the same order in both m-objects, e.g. in each m-object level *category* comes before level *model* (level order compatibility). Both m-objects associate attribute *taxRate* with level *category* (stability of attribute levels) and define domain *Integer* for this attribute (stability of attribute domains). If m-object *Product* defined a value for this attribute, the value would need to be *20* to ensure compatibility of attribute values.

M-objects do not only inherit levels and attributes from their parents, but can also introduce new levels and add attributes to levels. M-objects use names instead of numeric potencies to identify levels, which enables introducing additional levels without affecting existing navigation paths.

Example 5 (extensibility)**.** In Figure 2, m-object *Car* inherits levels *category*, *model* and *physical entity* from m-object *Product*. It adds level *brand* to define that cars are further categorized by their brand. Note that the additional level *brand* applies only to descendant m-objects of *Car* and not to other sub-hierarchies such as descendants of *Book*. Additionally, m-object *Car* extends level definitions of m-object *Product* by adding attribute *maxSpeed* to level *model* and attribute *mileage* to level *physical entity*.

By concretizing a set of m-objects, the m-objects form concretization hierarchies. There do not only exist consistency criteria for individual concretizations, but also for such hierarchies, which are explained in Definition 3.

Definition 3 (Consistent Concretization Hierarchy of M-Objects)**.** *A concretization hierarchy of a set of m-objects O is defined by an acyclic relation $H \subseteq O \times O$, which forms a forest (set of trees). Let $o, o' \in O$, then o' is said to be a* direct concretization of *o or*

o' concretises o, *if* $(o', o) \in H$, *and to be an* indirect concretization of o *if* $(o', o) \in H^+$. *A concretization hierarchy* H *of a set of m-objects* O *is* consistent, *iff*

1. *Each* $o \in O$ *is a m-object according to Definition 1.*
2. *For each pair of m-objects* $(o', o) \in H$, o' *is a consistent concretization of* o *according to Definition 2.*
3. *Each attribute and level is introduced at only one m-object:*
   (a) $a \in (A_o \cap A_{o'}) : \exists \bar{o} \in O : (o, \bar{o}) \in H \wedge (o', \bar{o}) \in H \wedge a \in A_{\bar{o}}$ *(*unique induction rule for attributes*)*
   (b) $l \in (L_o \cap L_{o'}) : \exists \bar{o} \in O : (o, \bar{o}) \in H \wedge (o', \bar{o}) \in H \wedge l \in L_{\bar{o}}$ *(*unique induction rule for levels*)*

Example 6 (concretization hierarchy)**.** In Figure 2, m-object *Porsche911* is a direct concretization of m-object *Car* and an indirect concretization of m-object *Product*. M-objects *Book* and *Car* have a common attribute *taxRate*. To be consistent, these m-objects must have a common ancestor in the concretization hierarchy, which also defines this attribute, namely m-object *Product* in the example.

Each m-object plays multiple roles with regard to the classical semantic abstraction hierarchies: (i) A m-object can be regarded as an aggregate object of its children m-objects. (ii) It represents for each level of direct or indirect descendants the class of descendant m-objects of that level. Therefore it plays multiple class roles in multiple generalization hierarchies. (iii) Concerning classification hierarchies, it plays multiple class and meta^n class (meta, meta-meta class, ...) roles: For a given n, where n is less than the number of levels of descendants, a m-object plays multiple meta^n class roles for various combinations of levels of descendants. We discuss this in the remainder of this section.

A m-object represents for each level of direct or indirect descendants the class of descendant m-objects of that level. The notion of class refers first to the common structure (often referred to as *type* of a class) and second to the set of descendant m-objects at a specific level (often referred to as extension or members of a class). To refer to the set of m-objects at level l beneath m-object o, we write $o\langle l \rangle$. For example, $car\langle model \rangle$ refers to the set of m-objects at level *model* beneath m-object *Car*.

Definition 4 (Class Extension)**.** *The class of m-objects of m-object* $o \in O$ *at level* $l \in L_o$*, denoted as* $o\langle l \rangle$*, is defined by*

$o\langle l \rangle \stackrel{\text{def}}{=} \{o' \mid (o', o) \in H^* \wedge \hat{l}_{o'} = l\}$.

Example 7 (Class extension)**.** M-object *Product* possesses the following classes:

Product⟨catalog⟩ = {Product}
Product⟨category⟩ = {Book, Car}
Product⟨model⟩ = {HarryPotter4, Porsche911CarreraS, Porsche911GT3}
Product⟨physical entity⟩ = {myCopyOfHP4, myPorsche911CarreraS}

A m-object does not only define a class for each of its levels, but also, implicitly, multiple meta^n-classes. In this paper we limit the discussion on meta^n-classes on their extensional role, i.e. a meta-class considered as a set of classes, and do not deal with the structural role of meta-classes (meta-types).

Definition 5 (Meta-Classes and their Extensions)**.**

1. *For each m-object* o *and each pair of levels* $(l_0, l_1) \in P_o^+$ *meta-class* $o\langle l_1\langle l_0 \rangle\rangle$ *is defined as the set of classes containing for each m-object* o' *that is a descendant of* o *at level* l_1 *class* $o'\langle l_0 \rangle$*, i.e.*
   $o\langle l_1\langle l_0 \rangle\rangle \stackrel{\text{def}}{=} \{o'\langle l_0 \rangle \mid o' \in o\langle l_1 \rangle\}$.
2. *For* $o \in O$ *and* $l_0 \in L_o$ *meta-class* $o\langle\langle l_0 \rangle\rangle$ *is the set of all classes containing m-objects at level* l_0 *and that are descendants of* o*, i.e.,*
   $o\langle\ \langle l_0 \rangle\rangle \stackrel{\text{def}}{=} \{o'\langle l_0 \rangle \mid (o', o) \in H^*\}$

Example 8 (meta-classes and their extension)**.** The following meta-classes are based on m-object *Product*, as given by Figure 2 (Singleton-meta-classes like Product⟨catalog⟨category⟩⟩ are omitted):

Product⟨category⟨model⟩⟩ = {Car⟨model⟩, Book⟨model⟩}
Product⟨category⟨physical entity⟩⟩ = {Car⟨physical entity⟩, Book⟨physical entity⟩}
Product⟨model⟨physical entity⟩⟩ = {HarryPotter4⟨physical entity⟩, Porsche911CarreraS⟨physical entity⟩, Porsche911GT3⟨physical entity⟩ }

Product⟨⟨model⟩⟩ = {Product⟨model⟩, Car⟨model⟩, Book⟨model⟩, Porsche911⟨model⟩}
Product⟨⟨physical entity⟩⟩ = {Product⟨physical entity⟩, Car⟨physical entity⟩, Book⟨physical entity⟩, HarryPotter4⟨physical entity⟩, Porsche911⟨physical entity⟩, Porsche911CarreraS⟨physical entity⟩, Porsche911GT3⟨physical entity⟩ }

Definition 6 (Meta^n-Classes and their Extensions)**.**

1. *For each m-object* o *and (n+1)-tuple of levels* $(l_0, l_1, \ldots, l_n)$*, where* $n>1$ *and for* $i=1..n : (l_{i-1}, l_i) \in P_o^+$*, meta^n-class* $o\langle l_n\langle l_{n-1} \ldots \langle l_0 \rangle \ldots\rangle\rangle$ *is defined as the following set of meta^{n-1}-classes:*
   $\{o'\langle^{n-1} l_{n-1} \ldots \langle^0 l_0 \rangle \ldots\rangle \mid o' \in o\langle l_n \rangle\}$.
2. *For each m-object* o *and each level* $l_0 \in L_o$ *meta^n-class* $(n > 1)$ *is the following set of meta^{n-1}-classes*
   $\{o'\langle^{n-1} l_{n-1} \ldots \langle^0 l_0 \rangle \ldots\rangle \mid (o', o) \in H^* \wedge (\forall i \in 1..n\text{-}1 : (l_{i-1}, l_i) \in P_{o'}^+)\}$

Example 9 (meta^n-classes and their extensions)**.** The following meta^2-classes are based on m-object *Product*, as given by Figure 2 (Singleton-meta^n-classes like Product⟨catalog⟨category⟨model⟩⟩⟩ are omitted):

Product⟨category⟨model⟨physical entity⟩⟩⟩ = {Car⟨model⟨physical entity⟩⟩, Book⟨model⟨physical entity⟩⟩}

Product⟨⟨⟨physical entity⟩⟩⟩ = {Product⟨category⟨physical entity⟩⟩, Product⟨model⟨physical entity⟩⟩, Car⟨brand⟨physical entity⟩⟩, Car⟨model⟨physical entity⟩⟩, Book⟨model⟨physical entity⟩⟩}

4 Multi-Level Relationships

In this section we introduce, exemplify and formally define multi-level relationships (m-relationships) as a high-level modeling primitive for modeling relationships at multiple levels of abstraction. We exemplify that, when using traditional modeling and constraint languages (e.g. UML with OCL), dependencies between different abstraction levels of a relationship have to be modeled explicitly and specifically for each such relationship. We therefore advocate the use of m-relationships as a generic alternative. We show how m-relationships connect m-objects at multiple levels, how they can be concretized, and we look

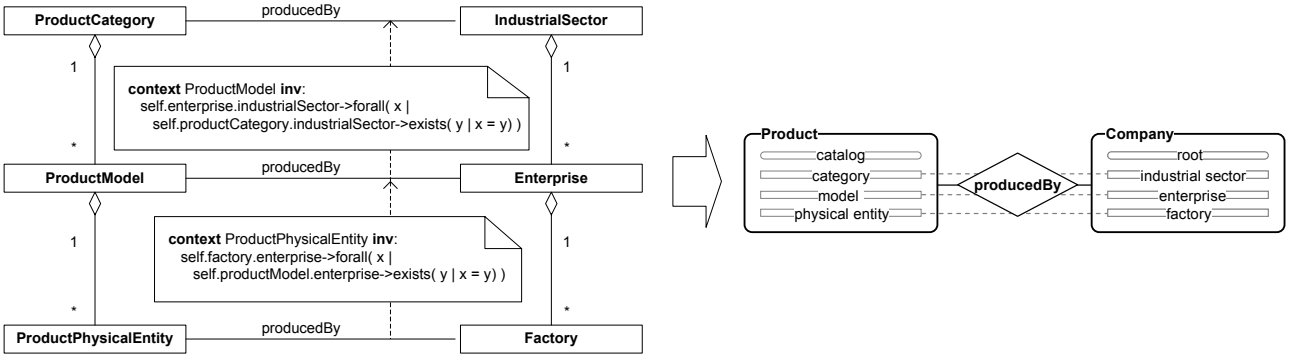

Figure 3: Basic Idea of Multi-Level Relationships (simplified), Left: UML and OCL, Right: M-Relationships

at the roles which they can play in concretizations. Finally, we define rules for consistent concretization of m-relationships. For simplicity, we consider only binary relationships in this paper and do not consider cardinality constraints.

So far we have shown how to reduce accidental complexity by describing domain-objects at multiple levels of abstraction through m-objects. Similarly, relationships occur at multiple levels of abstraction. We exemplify this below.

Example 10 (Sample Problem continued)**.** Consider our sample product catalog (Example 1). It describes products at three levels of abstraction (*physical entity*, *model* and *category*) and it describes the companies in which products are produced at three levels as well (*factory*, *enterprise*, and *industrial sector*). Relationship *producedBy* between levels *physical entity* and *factory* can be abstracted to *product model* and *enterprise* and further to *product category* and *industrial sector*. The following dependencies exist between these abstraction levels of relationship *producedBy*: (1) A *physical entity* can only be produced at a *factory* that belongs to an *enterprise* that produces the corresponding *product model*. (2) A *product model* can only be produced at an *enterprise* that belongs to an *industrial sector* that produces the corresponding *product category*. Specifically, product category *Car* is produced by industrial sector *CarManufacturer*, each model of car brand *Porsche911* is produced by *Porsche Ltd* and physical entity *myPorsche911CarreraS* is produced by factory *Porsche Zuffenhausen*.

Using standard UML (without OCL) one could specialize an association (e.g. producedBy between *PhysicalEntity* and *Factory*) for each specialization of the associated classes (e.g. *Porsche911PhysicalEntity* and *PorscheFactory*). But it is not possible to express implicitly through UML modeling primitives alone the above identified dependencies between the different abstraction levels of a relationship (e.g. that a *physical entity* can only be produced by a *factory* that belongs to an *enterprise* that produces the correspondent *product model*).

Dependencies between abstraction levels of a relationship could be expressed explicitly using a constraint language like OCL. Thereby the relationship has to be split into multiple associations (each representig one abstraction level). Then, the dependencies can be modeled by explicit and relationship-specific constraints that are imposed from a higher to a lower level of abstraction of the *producedBy* relationship (cf. left part of Figure 3). In HERM (Thalheim 2000) dependencies between relationships are defined more concisely by path inclusion constraints, but still have to be defined explicitly and relationship-specific.

The basic idea of m-relationships is to provide a high-level modeling primitive that (1) encapsulates different abstraction levels of a relationship (cf. right part of Figure 3), (2) implies extensional constraints between different abstraction levels of a relationship (as exemplified above), (3) supports heterogenous hierarchies (e.g. additional level *brand* at category *Car*), and (4) can be exploited for navigating and querying. This leads to application models that are easier to define, understand, and maintain.

The use of a high-level modeling primitive for m-relationships is favorable over modeling multiple associations with associated explicit constraints; such as modeling a composition in UML is favorable over modeling a simple association and defining its composition-semantics using OCL. This approach is in line with the aim of semanic data models to "provide high-level modeling primitives to capture the semantics of an application environment" (Hammer & McLeod 1981).

M-relationships are analogous to m-objects in that they describe relationships between m-objects at multiple levels of abstraction. M-relationships are bidirectional. To facilitate referencing objects involved in binary relationships, we take, however, a (potentially) arbitrary directional perspective by considering one object in the *source* role and the other object in the *target* role of the relationship. Each m-relationship links a source m-object with a target m-object. Additionally, it connects one or more pairs of levels of the source and the target. These connections between source- and target-levels constitute the abstraction levels of a m-relationship. They define that m-objects at source-level are related with m-objects at target-level. (Remember that a m-object is at a certain level if this level is its top level.) We note that the generic roles *source* and *target*, which we introduced here for simplicity, may be easily replaced by relationship-specific role names.

Definition 7 (M-Relationship)**.** *Let O denote a universe of m-objects. A m-relationship $r = (s_r, t_r, C_r)$ consists of two m-objects, $s_r \in O$ and $t_r \in O$, linked by m-relationship r. M-relationship r connects levels of source m-object s_r, $L_{s_r} \subseteq L$, to levels of target m-object t_r, $L_{t_r} \subseteq L$, as specified by $C_r \subseteq (L_{s_r} \times L_{t_r})$.*

We denote by R the set of m-relationships. Each m-relationship $r \in R$ has a name defined by function $n : R \rightarrow N$ where the name must be unique *only* between the m-objects it links.

Example 11 (m-relationship)**.** To model that car models have a designer, Figure 4 introduces a relationship *designedBy* between source m-object *Car* and target m-object *Person*. More precisely, it links source-level *model* of m-object *Car* to target-level *individual* of m-object *Person*. While relationship *designedBy* only links one source-level with one target-level, relationship *producedBy* between *Product* and

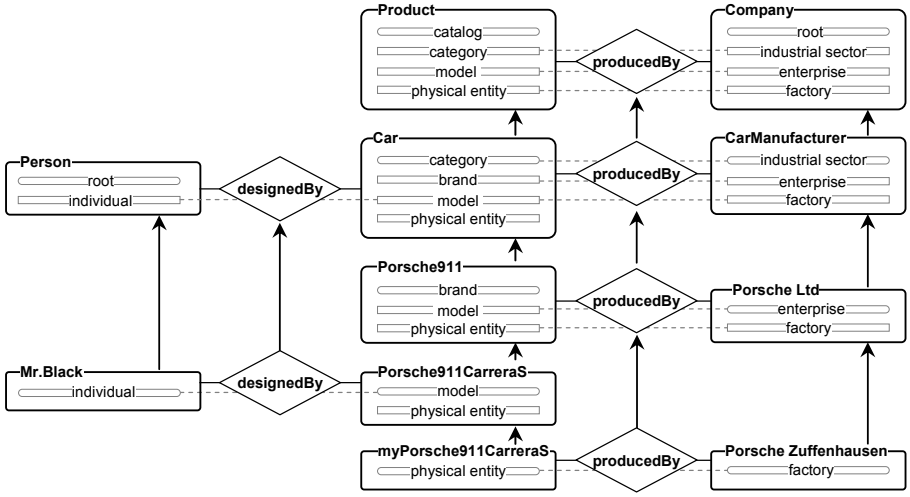

Figure 4: Product catalog modeled with m-objects and m-relationships

Company specifies multiple source- and target-levels. The relationship expresses that product categories are produced by industrial sectors, product models by enterprises and physical entities by factories.

Multiple pairs of source-target-levels express that the relationship exists at multiple levels of abstraction. To be consistent, a m-relationship must only specify source levels and target levels that exist in the source object and target object, respectively. Additionally, the relative order between source- and target-levels must be the same. Informally, this means that connections between source- and target-levels must not cross, a property which we call multi-level coherence.

Definition 8 (Multi-Level Coherence)**.** *A m-relationship r is* multi-level coherent *iff for* $(l_s, l_t) \in C_r, (l'_s, l'_t) \in C_r \rightarrow (((l_s, l'_s) \in P^*_{s_r} \wedge (l_t, l'_t) \in P^*_{t_r}) \vee ((l'_s, l_s) \in P^*_{s_r} \wedge (l'_t, l_t) \in P^*_{t_r}))$.

Example 12 (multi-level coherence)**.** Looking at m-relationship *producedBy* between *Car* and *CarManufacturer* of Figure 4, multi-level coherence is fulfilled because the relative order of pairs of source-target-levels (*category* to *industrial sector*, *model* to *enterprise*, *physical entity* to *factory*) complies with the relative order of target-levels (*industrial sector*, *enterprise*, *factory*) at target m-object *Company*.

Like m-objects, m-relationships can be concretized. Concretizing a m-relationship means to substitute its source or its target for a direct or indirect concretization. The concretizes-relationship between two m-relationships expresses instantiation and/or specialization.

Example 13 (concretization)**.** Consider m-relationship *designedBy* between m-objects *Car* and *Person* in Figure 4. M-relationship *designedBy* between m-objects *Porsche911CarreraS* and *Mr.Black* concretizes this m-relationship. In this case, the concretization solely represents an instantiation, i.e. no further concretization is possible. Now look at m-relationship *producedBy* between m-objects *Product* and *Company*. M-relationship *producedBy* between *Car* and *CarManufacturer* concretizes this m-relationship, expressing that cars can only be produced by car manufacturers. This concretization further defines that each car model must be produced by enterprises that belong to the car manufacturer sector. Similarly, each physical entity of cars must be produced by a factory of the car manufacturer sector.

Dependent on the levels which a m-relationship connects, it can play various roles:

- *Relationship class:* A relationship that links a non-top-level of the source m-object with a non-top-level of the target m-object can be regarded as a relationship class. It defines that m-objects at the source-level may or must — depending on whether the relationship is optional or mandatory, a distinction which we do not discuss in this paper — have a relationship with m-objects at the target-level, which are concretizations of the target m-object. A m-relationship that links n pairs of non-top-levels, with $n \geq 2$, can further be seen as a meta^{n-1} relationship class because these links constrain how this relationship has to be further concretized.
- *Relationship occurrence:* A m-relationship that links the top-level of the source m-object with the top-level of the target m-object can be seen as relationship occurrence.
- *Shared relationship:* A m-relationship that links a non-top-level with a top-level is shared with lower levels, i.e. it is applicable to m-objects at this non-top-level. It cannot be further concretized by m-objects that still possess that non-top-level, but only by more concrete m-objects beneath.

Example 14 (roles of m-relationships)**.** In Figure 4, the relationship *designedBy* between source *car model* and target *person individual* can be regarded as a relationship class, defining that each car model may or must (see above) have a relationship with an individual person. In our example, there is such a relationship between the car model *Porsche911CarreraS* and the individual person *Mr.Black*. This relationship represents a relationship occurrence. Similarly, relationship *producedBy* between *Product* and *Company* connects source-level *model* of m-object *Product* with target-level *enterprise* of m-object *Company*, which is a relationship class. There is no explicit relationship occurrence between m-object *Porsche911CarreraS* and m-object *Porsche Ltd*. But there is a shared relationship between m-objects *Porsche911* and *Porsche Ltd* at the respective levels. This relationship defines that each *Porsche911 model*, i.e. including *Porsche911CarreraS*, is produced by *Porsche Ltd*. It also takes the role of a relationship class, constraining the producers of physical entities of *Porsche911* to factories of *Porsche Ltd*.

The relationship cannot be further concretized by m-object *Porsche911CarreraS*, which still possesses the (former) non-top-level *model*, but only at m-object *myPorsche911CarreraS*.

M-relationships can be concretized like m-objects. This is achieved by concretizing source and target objects. For a consistent concretization of a m-relationship to a more concrete m-relationship, both relationships must connect the same levels with regard to their common levels.

Definition 9 (Consistent Concretization of M-Relationships)**.** *A m-relationship r' is a* consistent concretization *of m-relationship r iff*

1. *r' concretizes the source object or the target object of r, or concretizes both objects related by r:* $((s_{r'}, s_r) \in H^+ \wedge (t_{r'}, t_r) \in H^*) \vee ((s_{r'}, s_r) \in H^* \wedge (t_{r'}, t_r) \in H^+)$ *(*source and/or target concretization*)*
2. *r and r' connect the same source-target levels for the levels shared between its source and target objects:* $l_s \in (L_{s_r} \cap L_{s_{r'}}), l_t \in (L_{t_r} \cap L_{t_{r'}}) \rightarrow ((l_s, l_t) \in C_r \leftrightarrow (l_s, l_t) \in C_{r'})$ *(*stability of source-target levels*)*

Example 15 (consistent concretization)**.** M-relationship *producedBy* between *Car* and *CarManufacturer* is a consistent concretization of m-relationship *producedBy* between *Product* and *Company* in Figure 4. Its source *Car* concretizes *Product* and its target *CarManufacturer* concretizes *Company* (target and/or source concretization). Both relationships have the same pairs of source-target-levels (*category* to *industrial sector*, *model* to *enterprise*, *physical entity* to *factory*) and thus fulfill stability of source-target levels.

Like m-objects, m-relationships are organized in a concretization hierarchy, defined by an acyclic relation $H_R \subseteq R \times R$, which forms a forest (set of trees). A concretization hierarchy H_R of a set of m-relationships R is *consistent*, iff for each pair of m-relationships $(r', r) \in H_R$, r' is a consistent concretization of r.

5 Working with M-Objects

In this section, we show how to create and query m-objects and m-relationships, which can be achieved by extending existing data manipulation and query languages. As sample language, we briefly sketch multi-level SQL (M-SQL), which is a homogenous, light-weight extension to SQL-3, by examples. A full presentation of the language is beyond the scope of the paper. We first demonstrate how to create m-objects and m-relationships and then define *upward navigability*, and *navigation along m-relationships* as basis for querying m-objects and m-relationships.

To create multi-level models, M-SQL introduces an operator to create resp. concretize m-objects and m-relationships. Figure 5 exemplifies how to create m-object *Product* and its concretization *Car* using M-SQL. It also shows how to create the m-relationship *producedBy* between *Product* and *Company*, and its concretization to relationship *producedBy* between *Car* and *CarManufacturer*.

Upward navigation allows direct and stable access from a m-object o to its ancestor m-object at a specific level l, denoted by $o[l]$. For example, to navigate from m-object *myPorsche911CarreraS* to its ancestor at level *category*, i.e. *Car*, we write *myPorsche911CarreraS[category]*.

```
CREATE M-OBJECT Product
 (catalog (desc STRING),
  category (taxRate INTEGER),
  model (listPrice FLOAT),
  physicalEntity (serialNr STRING)),
 SET desc = "Our Products";

CREATE M-RELATIONSHIP producedBy
 BETWEEN Product AND Company
 CONNECTING (category TO industrialSector,
             model TO enterprise,
             physicalEntity TO factory);

CREATE M-OBJECT Car UNDER Product
 ADD LEVEL brand UNDER LEVEL category
                        (marketLaunch DATE),
 ALTER LEVEL model ADD (maxSpeed INTEGER),
 ALTER LEVEL physicalEntity ADD
                         (mileage INTEGER),
 SET category.taxRate = 20;

CREATE M-RELATIONSHIP producedBy BETWEEN
  Car AND CarManufacturer UNDER
  producedBy BETWEEN Product AND Company;
```

Figure 5: Creating m-objects *Product* and *Car* and m-relationships *producedBy* as shown in Figures 2 and 4

Definition 10 (Upward Navigation)**.** *The ancestor m-object of m-object $o \in O$ at level $l \in L_o$, denoted as $o[l]$, is defined by*

$o[l] \stackrel{\text{def}}{=} o' : (o, o') \in H^* \wedge \hat{l}_{o'} = l.$

Navigation along multi-level relationships is similar to dereferencing references in SQL-3. But, navigation in the context of concretization hierarchies of m-relationships is multi-faceted and expresses that all concretizations of a given m-relationship are to be traversed that lead to a m-object at some specified level.

Definition 11 (Navigation along m-relationships)**.** *The set of target m-objects (*source m-objects*) at level l reached by traversing a concretization of relationship r from source m-object (*from target m-object*, resp.) o, denoted as $o\text{->}r\langle l\rangle$ ($o\text{<-}r\langle l\rangle$, resp.), is defined by*

$o\text{->}r\langle l\rangle \stackrel{\text{def}}{=} \{o' \in O \mid \exists r' \in R : (r', r) \in H_R^* \wedge s_{r'} = o \wedge t_{r'} = o' \wedge \hat{l}_{o'} = l\}$

($o\text{<-}r\langle l\rangle \stackrel{\text{def}}{=} \{o' \in O \mid \exists r' \in R : (r', r) \in H_R^* \wedge t_{r'} = o \wedge s_{r'} = o' \wedge \hat{l}_{o'} = l\}$).

Class extension, upward navigation and navigation along m-relationships serve as basis for extending SQL-3's select-statement to support querying multi-level models. Identifiers of classes of m-objects of the form $o\langle l\rangle$ can be used where SQL-3 allows table names, primarily in the FROM-part of select statements. Upward navigation of the form $o[l]$ is allowed where SQL-3 allows dereferencing references, primarily in the select- and where-part of select statements. Navigation along m-relationships is allowed where dereferencing of references is expected.

```
SELECT c.maxSpeed, c.listPrice *
    (1 + c[category].taxRate)) AS grossPrice
FROM Car<model> c
WHERE c->Car-producedBy-CarManufacturer<enterprise>
                      = PorscheLtd;
```

Figure 6: M-SQL query retrieving information about car models

Example 16 (M-SQL query)**.** Figure 6 demonstrates how to query all car models (class *Car⟨model⟩*) that are produced by car manufacturer *PorscheLtd* (navigation along concretisations of m-relationship *producedBy* between *Car* and *CarManufacturer*, identified by `Car-producedBy-CarManufacturer`, to m-objects at level *enterprise*), retrieving their *maxSpeed* and their *grossPrice* (using upward navigation).

Variables can be used in place of object names in class or meta^n-class identifiers to refer to classes or meta^n-classes in queries. Such identifiers can be used wherever table names are expected in SQL-3.

```
1)SELECT m.name, c.taxRate, m.listPrice
  FROM Product<category> c, c<model> m
  WHERE c.taxRate > 15 AND m<physicalEntity>.COUNT > 10

2)SELECT mp.name, mp.COUNT
  FROM Product<model<physical entity>> mp
```

Figure 7: Querying members of members and meta-classes

Example 17 (Querying members of members)**.** Query 1 in Figure 7 retrieves the name, the tax rate and the list price of each product model that belongs to a product category with a *taxRate* higher than *15* and which have more than 10 members at level physical entity. Query 2 retrieves for m-object *Product* the size of each class of physical entities at level model, i.e. in our running example: (HarryPotter4⟨physical entity⟩,1), (Porsche911CarreraS⟨physical entity⟩,1), (Porsche911GT3⟨physical entity⟩,0).

6 Related Work

Several approaches have been presented in recent years to model domain objects at multiple levels of abstraction. The prevailing techniques are *materialization* (Goldstein & Storey 1994, Pirotte et al. 1994, Dahchour et al. 2002), *powertypes* (Odell 1998, Henderson-Sellers & Gonzalez-Perez 2005, Gonzalez-Perez & Henderson-Sellers 2006), and *potency-based deep instantiation* (Atkinson & Kühne 2001, Kühne & Schreiber 2007). Note that each approach has been developed with a different focus in mind and that therefore the approaches complement one another. In this section, we compare these approaches with regard to the requirements presented in Section 2. For a detailed comparison of these different approaches see (Neumayr & Schrefl 2008). We conclude with an overview of further related work. For a comprehensive introduction to meta-modeling and multi-level modeling we refer to (Olivé 2007).

To the best of our knowledge, modeling of relationships at multiple levels of abstractions, as presented in this paper, has not been studied so far. To some extent, type parameters provided by some object-oriented languages can be used as a work-around (for an overview see (Bruce 2002)). In the absence of a modeling primitive for m-relationships in traditional conceptual modeling, multiple associations and explicit constraints between these associations have to be used to represent relationships at multiple levels of abstraction. In UML, these dependencies can be defined using the object constraint language (OCL) (cf. Figure 3). Similarly, in the Higher-order Entity-Relationship Model (HERM) (Thalheim 2000), these dependencies can be defined using path inclusion constraints.

Materialization (Goldstein & Storey 1994, Pirotte et al. 1994, Dahchour et al. 2002) is a generic relationship type, which can be regarded as a special kind of aggregation (aggregation in its broader sense as used in this paper). Each materialization relationship connects two object classes, a more abstract with a more concrete one. In this respect, this approach supports multiple levels of abstraction. To add attributes to non-uniform objects (e.g. book vs. car), it provides a very flexible and powerful attribute propagation mechanism. Propagation of attribute definitions over more than one materialization level could be accomplished by composing attribute propagation types T1 and T3 (as sketched in (Neumayr & Schrefl 2008)), however this might harm readability of models. Materialization is not suited for adding additional levels to certain sub-hierarchies and it does not offer special support for relationships and querying.

Powertypes were introduced by Odell (Odell 1998) and are further investigated by Henderson-Sellers and Gonzalez-Perez (Henderson-Sellers & Gonzalez-Perez 2005, Gonzalez-Perez & Henderson-Sellers 2006), especially as a basis for software engineering methodology. The powertype approach also offers support for ontological multi-level modeling as discussed in this paper. A powertype describes common properties of direct subclasses of the class it is associated with. A cascaded setup of multiple powertypes can be used to describe more than one level beneath some class, i.e. common properties of indirect subclasses on a specific level of the generalization hierarchy. By specializing powertypes, this approach supports extensibility but does not completely avoid fragmentation and redundancy.

Ontological metamodeling with potency-based deep instantiation was introduced by Atkinson and Kühne (Atkinson & Kühne 2001). They also introduced the distinction between ontological and linguistic metamodeling (Atkinson & Kühne 2003). Later, Kühne extended this approach to programming with multi-level models (Kühne & Schreiber 2007). Deep instantiation supports unbound classification levels. Considering the examples given by Kühne in (Kühne & Schreiber 2007), they would model our sample problem by combining deep instantiation and subclassing. Deep instantiation enables to define attributes of objects not only in their class, but also in a meta^n-class. With regard to extensibility, it supports adding attributes, levels and relationships. However, new classification levels apply to all sub-hierarchies, which complicates modeling non-uniform sub-hierarchies. Concerning fragmentation, deep instantiation cannot avoid multiple unconnected model elements on different classification levels. While this approach supports relationships and their specialization, it does not consider multi-level relationships. As a work-around, similar results could be achieved by using type parameters as shown in (Kühne & Schreiber 2007).

In the field of database design and conceptual modeling, starting from the classical work on semantic data models (for an overview see (Schrefl et al. 1984, Hull & King 1987, Peckham & Maryanski 1988)), a lot of research has been centered around classification, generalization and aggregation. Concerning multiple classification levels, Klas and Schrefl provided an approach to deep instantiation for multi-level object-oriented databases (Klas & Schrefl 1995). Telos (Mylopoulos et al. 1990) and its successor ConceptBase (Jarke et al. 1995) allowed unbound classification levels. Troyer and Janssen embed schemas into Schema Object Types and discuss its implication on subtyping (Troyer & Janssen 1993).

7 Conclusion

Applications of m-objects and m-relationships are manifold. As information systems grow in size or previously independent systems are integrated across related domains, the need to handle levels of similar, yet non-uniform objects arises. This need arises not only for conceptual models as such, but also and especially for data warehouses. Data warehouses that aggregate data from multiple organisations or enterprises with heterogenous information systems have to deal with heterogenous dimension hierarchies. M-object hierarchies can be used to cope with these heterogenities. Take for example a sales cube, modeled using the Dimensional Fact Model (Golfarelli et al. 1998), with three dimensions, *product*, *location* and *date*, where the *product* dimension consists of two levels, namely *model* and *category*. Using the basic Dimensional Fact Model, it is not possible to define that some product categories have additional levels, like in our sample problem (see Example 1), products of category *car* have an additional level, namely *brand*. In order to support modeling of such heterogenous dimension hierarchies in an extended Dimensional Fact Model one can simply replace the dimension *product* by m-object *product* and reuse its heterogenous level structure.

Apart from domain and data warehouse modeling, m-objects and m-relationships can serve as a basis for corresponding extensions of object-oriented programming languages, object-oriented and object-relational databases as well as for customizing information systems.

M-objects and m-relationships constitute a novel approach to multi-level domain modeling, which supports modeling objects and relationships at multiple levels of abstraction. The concretization hierarchy of m-objects combines aspects of the different abstraction hierarchies *classification, generalization and aggregation*, which avoids fragmentation and redundancy in modeling. Extending m-objects along the concretization hierarchy enables to model non-uniform objects as well as non-uniform sub-hierarchies. M-relationships link m-objects at multiple levels of abstraction and can be specialized along the concretization hierarchy. Working with m-objects and m-relationships is supported by M-SQL, which extends SQL-3 to create and query multi-level models.

References

Atkinson, C. & Kühne, T. (2001), The essence of multilevel metamodeling, *in* M. Gogolla & C. Kobryn, eds, 'Proceedings of the 4^{th} International Conference on the UML 2000, Toronto, Canada', LNCS 2185, Springer Verlag, pp. 19–33.

Atkinson, C. & Kühne, T. (2003), 'Model-driven development: A metamodeling foundation.', *IEEE Software* **20**(5), 36–41.

Bruce, K. B. (2002), *Foundations of Object-Oriented Languages: Types and Semantics*, The MIT Press, Cambridge, Massachusetts.

Dahchour, M., Pirotte, A. & Zimányi, E. (2002), 'Materialization and its metaclass implementation.', *IEEE Trans. Knowl. Data Eng.* **14**(5), 1078–1094. 0605.

Goldstein, R. C. & Storey, V. C. (1994), 'Materialization', *IEEE Trans. Knowl. Data Eng.* **6**(5), 835–842.

Golfarelli, M., Maio, D. & Rizzi, S. (1998), 'The dimensional fact model: A conceptual model for data warehouses', *Int. J. Cooperative Inf. Syst.* **7**(2-3), 215–247.

Gonzalez-Perez, C. & Henderson-Sellers, B. (2006), 'A powertype-based metamodelling framework', *Software and System Modeling* **5**(1), 72–90.

Hammer, M. & McLeod, D. (1981), 'Database description with sdm: a semantic database model', *ACM Trans. Database Syst.* **6**(3), 351–386. 0605.

Henderson-Sellers & Gonzalez-Perez (2005), 'Connecting powertypes and stereotypes', *Journal of Object Technology* **4**, 83–96.

Hull, R. & King, R. (1987), 'Semantic database modeling: survey, applications, and research issues', *ACM Comput. Surv.* **19**(3), 201–260.

Jarke, M., Gallersdörfer, R., Jeusfeld, M. A. & Staudt, M. (1995), 'Conceptbase - a deductive object base for meta data management.', *J. Intell. Inf. Syst.* **4**(2), 167–192.

Klas, W. & Schrefl, M. (1995), *Metaclasses and Their Application - Data Model Tailoring and Database Integration*, Springer.

Kühne, T. & Schreiber, D. (2007), Can programming be liberated from the two-level style: multi-level programming with deepjava, *in* R. P. Gabriel, D. F. Bacon, C. V. Lopes & G. L. S. Jr., eds, 'OOPSLA', ACM, pp. 229–244.

Mylopoulos, J., Borgida, A., Jarke, M. & Koubarakis, M. (1990), 'Telos: Representing knowledge about information systems', *ACM Trans. Inf. Syst.* **8**(4), 325–362.

Neumayr, B. & Schrefl, M. (2008), Comparison criteria for ontological multi-level modeling, Technical Report 08.03, Department of Business Informatics - Data & Knowledge Engineering, Johannes Kepler University Linz, Austria.
URL: *http://www.dke.jku.at/papers/TR0803.pdf*

Odell, J. J. (1998), *Advanced Object-Oriented Analysis & Design Using UML (also published as James Odell: Power Types. JOOP 7(2): 8-12 (1994))*, Cambridge University Press, chapter Power Types, pp. 23–32.

Olivé, A. (2007), *Conceptual Modeling of Information Systems*, Springer.

Peckham, J. & Maryanski, F. (1988), 'Semantic data models', *ACM Comput. Surv.* **20**(3), 153–189.

Pirotte, A., Zimányi, E., Massart, D. & Yakusheva, T. (1994), Materialization: A powerful and ubiquitous abstraction pattern., *in* J. B. Bocca, M. Jarke & C. Zaniolo, eds, 'VLDB', Morgan Kaufmann, pp. 630–641. 0605.

Schrefl, M., Tjoa, A. M. & Wagner, R. (1984), Comparison-criteria for semantic data models, *in* 'ICDE', IEEE Computer Society, pp. 120–125.

Thalheim, B. (2000), *Entity-Relationship Modeling: Foundations of Database Technology*, Springer.

Troyer, O. D. & Janssen, R. (1993), On modularity for conceptual data models and the consequences for subtyping, inheritance & overriding, *in* 'ICDE', IEEE Computer Society, pp. 678–685.