Formal Executable Theory of Multilevel Modeling

Mira Balaban[1(✉)], Igal Khitron[1], Michael Kifer[2], and Azzam Maraee[1]

[1] Computer Science Department, Ben-Gurion University of the Negev, Beersheba, Israel
{mira,khitron,mari}@cs.bgu.ac.il

[2] Computer Science Department, Stony Brook University, Stony Brook, NY, USA
kifer@cs.stonybrook.edu

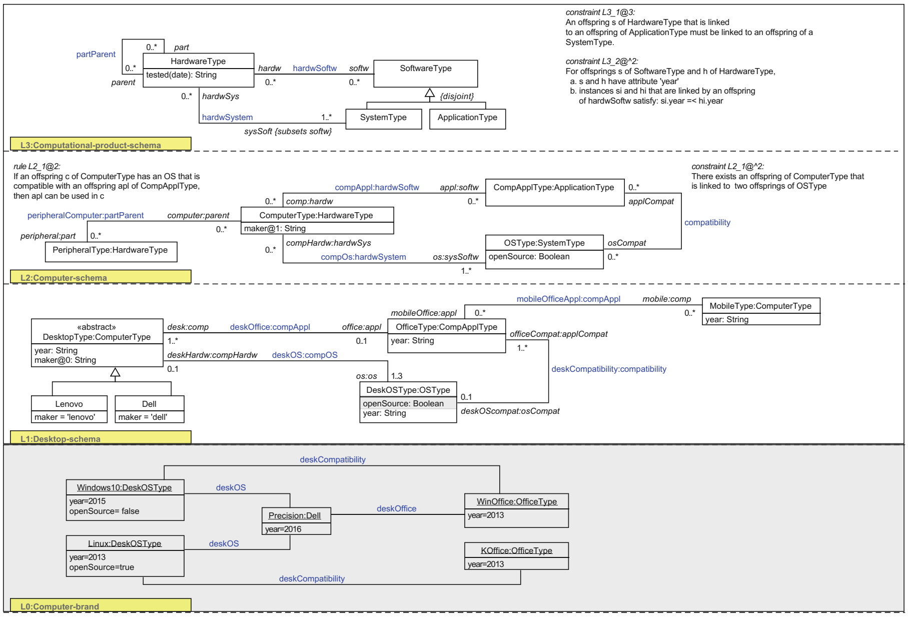
Abstract. Multi-Level Modeling (MLM) conceptualizes software models as layered architectures of sub-models that are inter-related by the instance-of relation, which breaks monolithic class hierarchies midway between subtyping and interfaces. This paper introduces a formal theory of MLM, rooted in a set-theoretic semantics of class models. The MLM theory is validated by a provably correct translation into the FOML executable logic. We show how FOML accounts for inter-level constraints, rules, and queries. In that sense, FOML is an organic executable extension for MLM that incorporates all MLM services. As much as the page budget permits, the paper illustrates how multilevel models are represented and processed in FOML.

Keywords: Multi-level modeling · Herbrand semantics · Class facet
Object facet · Executable logic

1 Introduction

Multilevel system modeling (MLM) views the enterprise as a layered collection of models that are inter-related by the *instance-of* (or *membership*) relation among objects and classes. The grounds for this approach are both philosophical and pragmatic. On philosophical grounds, researchers have been arguing that faithful modeling of real world domains cannot be restricted by the standard two-layer architecture of the OMG meta-modeling approach. They claim that natural domains have multiple levels of classification, and an artificial restriction to two layers yields models that are too weak [4,15]. On pragmatic grounds, researchers have argued that a multilevel architecture of models simplifies the management and evolution of complex software [23].

An important advantage of multilevel models is that in a monolithic class hierarchy structure every change affects the entire hierarchy. The conventional approach in software systems is to break monolithic hierarchies using interfaces



constraint L3_1@3:
An offspring s of HardwareType that is linked
to an offspring of ApplicationType must be linked to an offspring of a
SystemType.

constraint L3_2@^2:
For offsprings s of SoftwareType and h of HardwareType,
a. s and h have attribute 'year'
b. instances si and hi that are linked by an offspring
of hardwSoftw satisfy: si.year =< hi.year

rule L2_1@2:
If an offspring c of ComputerType has an OS that is
compatible with an offspring apl of CompApplType,
then apl can be used in c

constraint L2_1@^2:
There exists an offspring of ComputerType that
is linked to two offsprings of OSType

Fig. 1. A multilevel model of computer products

and delegation. In contrast, in MLM, class hierarchies are broken by the instance-of relation — midway between subtyping and interfaces. Thus, multiple modeling layers provide modular representation with controlled limited inheritance.

Figure 1 describes a multilevel model of a computational product. The model is split into disjoint layers L0, L1, L2, L3, where L0, `Computer-brand`, describes desktops brands with their hardware and software components; level L1 is `ComputerKind-schema`, the domain of computer kinds, their hardware and software; level L2 is `Computer-schema`, and level L3 describes the `Computational-Product-schema` for hardware and software components. Level L0 is an *object model* that serves as a partial instance for the *class model* at level L1. (L0 is shaded to indicate that it is not a class model and is unlike L1–L3 in this respect.) In a sense to be detailed in Sect. 2, L1 itself also serves as a partial instance for the class model at level L2. Similarly, L2 serves as an instance for the class model at level L3.

Each layer L_i except the first and the last plays a dual role: as an instance of the class model at level $i+1$ — the *object facet* role, and as a class model at level i — the *class facet* role. In our example, L0 has only the role of an object facet for level 1 (as an instance for L1), L3 has only the class facet role (for level 3). In general, adjacent layers in the multilevel architecture are related by the standard *instance-of* relationship between a class model and its instance object models, while within each class model layer, classes are organized via subclass and general association relationships. Hence, some classes and associations in layer i play the role of objects and links for the class model in layer $i + 1$. This is denoted using the standard notation $o : C$. Such classes are called *clabjects* [4,23], i.e., ***cla***sses that are also o***bject***s, and such associations are called *assoclinks*, i.e., ***asso***ciations that are also ***link***s. For example, in level L1, the clabjects are `DesktopType`, `OfficeType`, `CompOSType` and `MobileType`, and the assoclinks are `deskOffice`, `deskOS`, `mobileOfficeAppl` and `deskCompatibility`. The clabjects and assoclinks of *ComputerKind-schema* in level L1 form a partial legal instance of the class model *Computer-schema* in level L2. Properties (association-ends) of assoclinks are *renamed* when moving up a layer, denoted with the object instantiation notation: $level_i prop : level_{i+1} prop$, as in *desk* : *comp* in level L1.

The tradition of multilevel modeling enables assignment of a *potency* numerical attribute to elements of the model [24,28]. The intuition is that the potency of an element specifies the maximum number of allowed consecutive instantiations, which can be smaller than the level of the element. Potency 0 for a class (or association) means that it is not instantiated by a clabject (assoclink) in the immediate lower level. An attribute with potency 0 is not inherited by instance clabjects of the class of the attribute. The default potency of an element is its level, and is not noted explicitly in the diagram. In Fig. 1, potency is denoted with the "@n" sign. Attribute `maker` of class `ComputerType` in level L2 has potency 1 and, indeed, its instantiation ends in level L1. Gray background for an attributes indicates that they are inherited (cf. *openSource* in L1).

The schemas in a multilevel model can be constrained and extended by inter-layer *constraints* and *inference rules* (so called "deep" constraints and rules). These constraints often involve the notion of an *offspring*. An offspring of class C is any class or object that is connected to C by a chain of *instance-of* and *subclass* relationships. Figure 1 includes one *deep rule* and three *deep constraints*; they are specified at some levels and affect several layers below them. *Computational-product-schema* includes the constraint *L3_1* of potency 3. This means that it affects the schemas at levels L2, L1, and L0. It states that for an offspring h of `HardwareType`, if h is linked to an offspring of `ApplicationType`, then it must also be linked to some offspring of `SystemType`. Indeed, the `MobileType` clabject at L1 does not satisfy this constraint, and *ComputerKind-schema* is a partial instance of *Computer-schema*, that can be complemented into a legal instance that satisfies this constraint.

Single-potency [28] of degree n is a different kind of constraint, denoted with "@ˆn" sign: it is specified at some level i but constrains the level $i - n$. Typically this is done for convenience, as specifying a constraint at a higher level can be more succinct. For instance, in Fig. 1, the constraint *L*3_2@ˆ2 is specified at level L3, but applies to L1-classes *DesktopType*, *MobileType*, *OfficeType*, and *DeskOSType* — the offsprings of the L3-classes *HardwareType* and *SoftwareType* that are connected to these L3-classes by chains of *instance-of* relationships of length 2. One could have specified the constraint *L*2_2@ˆ2 directly at level L1, but then it would have to be repeated for each of the above four offsprings.

Inference rules provide a powerful representation mechanism that can derive intensional information that is not explicit. Rule *L2_1* has potency 2 and states that if a computer object c is an offspring of `ComputerType`, and has an operating system, which is compatible with an application a, then a is an application of c. With this rule, we can *derive* an L0-level link of the L1-level association `deskOffice` that links the L0-objects `KOffice` and `Precision`.

The overall architecture of a multilevel model organizes schemas in a layered architecture called an *ontological dimension*. Within each schema, the subclass relation is used, while the instance-of relation is used between adjacent schemas. A multilevel model can include several *overlapping* ontological dimensions. Due to space limitation, our formalization deals only with a single ontological dimension. Traditionally, MLM refers to the *linguistic modeling dimension* as a syntactic definition of models. This aspect is covered in Sect. 2.

The main contribution of this paper is a novel formal executable theory for MLM based on a model-theoretic semantics of class models [8,9]. Class models form multilevel models with the help of *Herbrand instances*, in which the syntactic symbols comprise the semantic domain. Our formal theory relies on *direct semantics* for multilevel models and accounts for complex model interactions.

The second contribution is a provably correct translation of the aforesaid MLM theory into the executable logic language FOML [7], which is based on an underlying theory of unrestricted chains of instance-of and subtyping relationships, combined with path expressions.[1] FOML enables direct, seamless encoding of the MLM theory, and is inspired by the industrial multilevel *Ink* project [1].

The third contribution has to do with the correctness of multilevel models and their intra-level and inter-level interactions. We define *consistency* and *finite-satisfiability* as natural extensions of their class model analogies, and show that intra-layer correctness can be checked for each schema independently. Moreover, FOML is capable of validating deep inter-level MLM dependencies.

Paper organization: Sect. 2 introduces a formal abstract syntax and model-theoretic semantics for multilevel models. Section 3 deals with correctness analysis Sect. 4 describes the encoding in FOML, reasoning and proving correctness, Sect. 5 discusses related work and Sect. 6 concludes the paper.

2 Multilevel Models–A Set-Theoretic Formalization

A multilevel model is a finite collection of *ontological dimensions*, each being a sequence of *schemas*. A schema consists of a *Herbrand instance* and a *class model*. The schemas in a dimension are defined over a global, sorted, infinite vocabulary $\mathcal{V} = \langle \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{A}, \rangle$ of *object* ($\mathcal{O}$), *class* ($\mathcal{C}$), *property* ($\mathcal{P}$), and *association* ($\mathcal{A}$) symbols. (Due to page limitation, we omit attributes, qualifiers, datatypes, and some constraints.) The sets $\mathcal{O}$ and $\mathcal{C}$ may overlap, and all other sets are disjoint. We define schemas, and combine them into an overall theory of multilevel models.

[1] Other languages, e.g., Telos [26] and RDF [22], also support these relationships.

2.1 Class Models: Abstract Syntax and Set-Theoretic Semantics

Abstract Syntax:[2] A *class model* over a global sorted vocabulary $\mathcal{V} = \langle \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{A} \rangle$ is a tuple $CM = \langle \mathcal{C}_{CM}, \mathcal{P}_{CM}, \mathcal{A}_{CM}, Mappings, Constraints \rangle$, where $\mathcal{C}_{CM} \subseteq \mathcal{C}$, $\mathcal{P}_{CM} \subseteq \mathcal{P}, \mathcal{A}_{CM} \subseteq \mathcal{A}$ are finite sets of class, property and association symbols, respectively, and *Mappings* and *Constraints* are defined below.
Mappings:

- $inverse : \mathcal{P}_{\mathcal{CM}} \rightarrow \mathcal{P}_{\mathcal{CM}}$ is a bijection that assigns to *every* property p its unique inverse, denoted p^{-1}, such that $inverse(p) \neq p$, and $(p^{-1})^{-1} = p$.
- $props \colon \mathcal{A}_{\mathcal{CM}} \rightarrow 2^{\mathcal{P}_{\mathcal{CM}}}$ is an injection where $props(a) = \{p, p^{-1}\}$ and every property from $\mathcal{P}_{\mathcal{CM}}$ appears in exactly one $props(a)$. If $p \in \mathcal{P}_{\mathcal{CM}}$, $assoc(p)$ denotes the association such that $p \in props(assoc(p))$. In the *Computational-Product* schema (level L3, Fig. 1), $props(hardwSoftw) = \{hardw, softw\}$, $softw = hardw^{-1}$, and $assoc(softw) = assoc(hardw) = hardwSoftw$.
- $source \colon \mathcal{P}_{\mathcal{CM}} \rightarrow \mathcal{C}_{\mathcal{CM}}$ and $target \colon \mathcal{P}_{\mathcal{CM}} \rightarrow \mathcal{C}_{\mathcal{CM}}$ are mappings such that $target(p) = source(p^{-1})$. They define the source and target classes of a property. For $a \in \mathcal{A}_{\mathcal{CM}}$, if $props(a) = \{p_1, p_2\}$ and $target(p_i) = C_i$, then $classes(a) = \{C_1, C_2\}$. In the aforesaid *Computational-Product* schema, $target(softw) = source(hardw) = SoftwareType, source(softw) = target(hardw) = HardwareType$, and $classes(hardwSoftw) = \{HardwareType, SoftwareType\}$.

Constraints: Class model constraints include *property multiplicities, association-classes, aggregation/composition, class hierarchy, generalization-set, property subsetting, redefinition, union, association-hierarchy, association-class hierarchy* and *xor*. We discuss only the following three:

- **Multiplicity mappings**: $min \colon \mathcal{P}_{\mathcal{CM}} \rightarrow \mathbb{N} \cup \{0\}$ and $max \colon \mathcal{P}_{\mathcal{CM}} \rightarrow \mathbb{N} \cup \{*\}$ assign minimum and maximum multiplicities to property symbols so that $min(p) \leq max(p)$ ($*$ denotes positive infinity).
- **Class hierarchy**: is an acyclic binary relation on class symbols in $\mathcal{C}_{CM}$: $C_2 \prec C_1$, means that C_2 is a subclass of C_1. The relation $\prec^+$ is a transitive closure of $\prec$ and $C_2 \preceq^* C_1$ means $C_2 = C_1$ or $C_2 \prec^+ C_1$. In the *Desktop* schema in Fig. 1, *Dell* is a subclass of *DesktopType*, i.e., $Dell \prec DesktopType$.
- **Property subsetting (subproperties)**: is an acyclic binary relation $\prec$ on property symbols:[3] $p_1 \prec p_2$ says that p_1 subsets (is a subproperty of) p_2. It is also required that (i) $source(p_1) \prec^* source(p_2)$, (ii) $target(p_1) \prec^* target(p_2)$, and (iii) $max(p_1) \leq max(p_2)$. In the *Computational-Product* schema of Fig. 1, $sysSoft \prec softw$ means that a *SystemType* which is a *sysSoft* of a *Hardware-Type* element, is also a *softw* of this element.

Compact Symbolic Notation for Associations: It is often convenient to have a compact notation that shows an association along with its properties,

[2] A full formalization of the UML class model, appears in [9].
[3] $\prec$ is overloaded for subproperties and subclasses.

classes and multiplicities. We write $a(C_1 \overset{p_1}{\underset{m_1..M_1}{}} \!\!\!\!\!\!\!\!\!\!\xrightarrow[]{} \overset{p_2}{\underset{m_2..M_2}{}} C_2)$—or $a(C_1 \overset{p_1 \quad\quad p_2}{\text{———}} C_2)$, if multiplicities are irrelevant—to denote an association a such that $props(a) = \{p_1, p_2\}$, $target(p_i) = C_i$, $min(p_i) = m_i$ and $max(p_i) = M_i$. For instance, the compact notation for association `hardwSoftw` in the *Computational-Product* schema of Fig. 1, is $hardwSoftw(HardwareType \overset{hardw \qquad softw}{\underset{0..* \qquad 0..*}{\text{————}}} SoftwareType)$.

Semantics: The standard set-theoretic semantics of class models associates such models with *instances* I, which consist of a semantic *domain* and a *denotation mapping* "$\cdot^I$" that assigns meaning to syntactic elements. Given a class model $CM = \langle \mathcal{C}_{CM}, \mathcal{P}_{CM}, \mathcal{A}_{CM}, Mappings, Constraints \rangle$: (1) each class symbol $c \in \mathcal{C}_{CM}$ is mapped to a set c^I of objects in the domain, called the *extension of* c; (2) each association symbol $a \in \mathcal{A}_{CM}$ is mapped to a relationship a^I, called the *extension* of a, between the extensions of the classes of a, i.e., $classes(a)$; (3) each property symbol $p \in \mathcal{P}_{CM}$ is mapped to a multivalued function $p^I : source(p)^I \rightarrow target(p)^I$, as follows: If $assoc(p) = a$ then for each $e \in source(p)^I$, $p^I(e) = \{e' \mid (e, e') \in a^I\}$, i.e., p^I is the projection of a^I on $source(p)^I$.

An *object* of a class model CM with respect to an instance I is an element in the domain of I that belongs to the extension of some class. The set of *objects* of CM with respect to I, denoted $objs_I(CM)$, is the union of all extensions in I of classes of CM, i.e., $\cup_{c \in \mathcal{C}_{CM}} c^I$. A *link* of CM with respect to an instance I is an element of the extension of some association a. A link includes a pair of objects $\mathsf{o}_1, \mathsf{o}_2$ in the domain of I, an identifier of the association to which it belongs, and the property roles of the above two objects. We denote links as $a(\mathsf{o}_1 \overset{\mathsf{p}_1 \quad \mathsf{p}_2}{\text{———}} \mathsf{o}_2)$, i.e., as labeled edges between nodes o_1, o_2, where $props(a) = \{p_1, p_2\}$, and $\mathsf{o}_1 \in p_1^I(\mathsf{o}_2), \mathsf{o}_2 \in p_2^I(\mathsf{o}_1)$. For an association symbol $a \in \mathcal{A}_{CM}$, $links_I(a) \overset{def}{=} \{a(\mathsf{o}_1 \overset{\mathsf{p}_1 \quad \mathsf{p}_2}{\text{———}} \mathsf{o}_2) \mid (\mathsf{o}_1, \mathsf{o}_2) \in a^I, \mathsf{o}_1 \in p_1^I(\mathsf{o}_2), \mathsf{o}_2 \in p_2^I(\mathsf{o}_1)\}$. The set $links_I(CM)$ of *all* links in CM with respect to I is the union of all links in I of all associations of CM, $\cup_{a \in \mathcal{A}_{CM}} links_I(a)$.

Legal and Herbrand Instances: An instance I of a class model is *legal*, denoted $I \models CM$, if its denotation mapping satisfies the class model constraints:

- **Multiplicity:** $min(p) \leq |p^I(\mathsf{e})| \leq max(p)$ for each $\mathsf{e} \in source(p)^I$.
- **Class-hierarchy:** If $C_1 \prec C_2$ then ${C_1}^I \subseteq {C_2}^I$.
- **Property subsetting:** If $p_1 \prec p_2$ and $\mathsf{e} \in source(p_1)^I$ then ${p_1}^I(\mathsf{e}) \subseteq {p_2}^I(\mathsf{e})$.

A *partial instance* is one whose denotation mapping can be extended to yield a legal instance. A *Herbrand instance*[4] of a class model CM over a global vocabulary $\mathcal{V} = \langle \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{A} \rangle$ is an instance of CM over the domain $\mathcal{O}$. Herbrand instances are often written using set notation, that explicitly lists the object members (from $\mathcal{O}$) of classes and the object pairs for associations: $\langle (\{o_1^i, \ldots, o_{n_i}^i\} = \mathcal{C}_i)^{C_i \in \mathcal{C}_{CM}},\ (\{(o_1^i, u_1^i), \ldots, (o_{n_i}^i, u_{n_i}^i)\} = a_i)^{a_i \in \mathcal{A}_{CM}} \rangle$. This writing saves explicit specification of property mappings and of empty extensions.

[4] By analogy with Herbrand interpretations in classical logic.

Example 1. *As explained in the introduction, levels L0–L2 in Fig. 1 play the role of instances for the class models at levels L1–L3. The notion of Herbrand instances formally captures this very idea. Specifically, L0 corresponds to the following Herbrand instance H_1 for the class model at level L1:* $H_1 = \{\ \{Windows10, Linux\} = DeskOSType,\ \{WinOffice, KOffice\} = OfficeType,\ \{Precision\} = Dell,\ \{(Precision, WinOfffice)\} = deskOffice,\ \{(Windows10, Precision), (Linux, Precision)\} = deskOS, \{(Windows10, WinOfffice), (Linux, KOfffice)\} = deskCompatibility\ \}$.
H_1 is a partial (legal) instance since `KOffice` *is an instance of* `OfficeType` *but has no* `deskOffice` *link.*

By analogy, we can construct a legal Herbrand instance H_2 for the class model at level L2. Unlike H_1, it is a complete *legal instance of this class model. However, when constructing a legal instance H_3 for the class model at level L3 out of the L2 level class model, we again get only a* partial *instance, as shown next:* $H_3 = \{\ \{ComputerType, PeripheralType\} = HardwareType,\ \{CompApplType\} = ApplicationType,\ \{OSType\} = SystemType,\ \{CompApplType, OSType\} = SoftwareType,\ \{(ComputerType, CompApplType)\} = hardwSoftw,\ \{(ComputerType, OSType)\} = hardwSystem\}$.
H_3 is a partial *Herbrand instance for level L3 because the subsetting constraint $sysSoft \prec softw$ in level L3 implies that the extension $hardwSoftw^{H_3}$ must include the extension of $hardwSystem^{H_3}$ but, by the above, it does not.* □

Semantic Relationships: An instance I of a class model CM is *empty* if all class extensions are empty and it is *infinite* if some class extension is infinite. A class model constraint *constr* is *entailed* by a class model CM, denoted $CM \models constr$, if it holds in every legal instance of CM. A class model CM_2 is *entailed* by class model CM_1, denoted $CM_1 \models CM_2$, if every legal instance of CM_1 is a legal instance of CM_2. Class models are *equivalent*, denoted $CM_1 \equiv CM_2$, if they have the same set of legal instances.

The following claim says that, for reasoning purposes, it is sufficient to consider legal Herbrand instances only.

Claim. For a class model CM and a class model constraint *constr*, *constr* holds in every legal Herbrand instance of CM, denoted $CM \models_H constr$, if and only if it is *entailed* by CM, i.e., $CM \models constr$.

2.2 Multilevel Models: Abstract Syntax and Model-Theory

First we introduce the notion of *mediation*, which connects a class model CM to an immediately higher class model CM'.

Definition 1. *Given a pair of class models over the same vocabulary:* $CM = \langle \mathcal{C}_{CM}, \mathcal{P}_{CM}, \mathcal{A}_{CM}, Mappings, Constraints\rangle$, *and* $CM' = \langle \mathcal{C}_{CM'}, \mathcal{P}_{CM'}, \mathcal{A}_{CM'}, Mappings', Constraints'\rangle$. *Let H' be partial Herbrand instance of CM'. Then, H' is a Herbrand mediator of type (CM, CM') if the objects and links of H' are classes and associations of CM, respectively. Formally:*

1. *$objs_{H'}(CM') \subseteq \mathcal{C}_{CM}$;*
2. *There is a 1:1 mapping link2assoc from $links_{H'}(CM')$ to $\mathcal{A}_{CM}$, that satisfies: link2assoc($a'(\mathtt{o}_1 \xrightarrow{\mathtt{p}'_1 \quad \mathtt{p}'_2} \mathtt{o}_2)$) $= a$, where $a \in \mathcal{A}_{CM}$ and $classes(a) = \{o_1, o_2\}$. The mapping link2assoc extends to properties as follows: link2assoc(p'_i) $= p_i$, assuming $props(a) = \{p_1, p_2\}$ and $target(p_i) = o_i$. That is, link2assoc maps links in H' to associations of CM, turning end-objects into end-classes and renaming association and properties.*

The partiality of the Herbrand mediator implies that not all elements in a higher level must be instantiated in a lower level. The *potency* assignment, introduced below, is an explicit mechanism for controlling inter-level instantiation.

Abstract Syntax: A multilevel model over a global sorted vocabulary $\mathcal{V} = \langle \mathcal{O}, \mathcal{C}, \mathcal{P}, \mathcal{A} \rangle$ is a finite set $\{\Theta^i\}_{1 \leq k \leq m}$, of *ontological* modeling dimensions. Each modeling dimension is a finite sequence of *schemas* $S_1, ..., S_n$, where each S_i is a pair $\langle H_i, CM_i \rangle$. The CM_i components are *class models* over $\mathcal{V}$; for $1 < i \leq n$, the H_i components are *Herbrand mediator*s of type (CM_{i-1}, CM_i); H_1 is a partial legal Herbrand instance of CM_1. The sets of classes in S_is are pairwise disjoint.

Returning to Fig. 1, Example 1 shows the legal partial instances H_1, H_2, H_3 for the class models L1, L2, L3 and illustrates how these instances were constructed out of the lower-level models L0, L1, and L2. The ontological modeling dimension depicted in the figure if formally represented as a sequence of these schemas: $\langle H_1, \mathtt{L1} \rangle$, $\langle H_2, \mathtt{L2} \rangle$, $\langle H_3, \mathtt{L3} \rangle$.

Semantics: A *legal instance* of a multilevel ontological dimension $(S_i = \langle H_i, CM_i \rangle)_{1 \leq i \leq n}$ is a sequence of Herbrand instances $(H'_i)_{1 \leq i \leq n}$, such that each each H'_i is a legal Herbrand instance of CM_i and, for all $i \geq 1$, H'_i includes H_i. That is, H'_i extends the partial Herbrand instance H_i into a full legal instance of CM_i.

Potency Assignment: The tradition of multilevel modeling enables assignment of a *potency* numerical attribute to elements of the model [24,28]. The intuition is that the potency of an element specifies the maximum number of allowed consecutive instantiations, which can be smaller than the level of the element[5]. We define the potency of elements in two steps:

(i) Let $\mathcal{C}_i, \mathcal{A}_i$ be the sets of classes and associations at level i. For each i, the partial function *user_defined_potency*: $\mathcal{C}_i \cup \mathcal{A}_i \rightarrow \mathbb{N}$ assigns a natural number $\leq i$ to some classes and associations, subject to *consistency requirements* below.

(ii) user_defined_potency is extended to a total *potency* function, which is obtained by propagating the values of *user_defined_potency* and using the *level* values as the default. Due to page limitation, we define *potency* for classes only.

1. For each class C' in a class model CM_i such that *user_defined_potency*(C') has a value, set *potency*(C') = *user_defined_potency*(C').

[5] Other known forms of potency are constraints on off-springs instead of instantiations.

2. For a class C' in CM_i for which $potency(C') = k$ where $k > 0$ and a class C in CM_{i-1} (i.e., $C \in C'^{H_i}$), set $potency(C) = k - 1$.
3. For every class C in a class model CM_i, $i > 0$, for which $potency(C)$ is undefined, set $potency(C) = level(C)$.

A potency function is ***inconsistent*** if it assigns more than one value to some class, or if a 0-potency class has direct instances. Formally:

- *Direct instantiation of a 0-potency class*: Let C' be a class in a class model CM_i (level i) with $potency(C') = 0$. The potency function is inconsistent if C' has a direct instance (in level $i-1$), i.e., if there is a class C in CM_{i-1} in level $i-1$ ($C \in C'^{H_i}$) with no intermediate class C'' (in level i) between C and C' (i.e., $C'' \prec C' \in CM_i$ and $C \in C'^{H_i}$).
- *Contradiction*: $potency(C)$ has more than one value for some clabject C. This may happen if $user\_defined\_potency$ is over-specified, for example in the presence of multiple inheritance.

3 Analysis of Multilevel Models

Correctness of a multilevel model depends on *intra-level* correctness, which refers to the correctness of each class model and its object facet, and on *inter-level* correctness, which is determined by deep constraints and their interaction with the class models they constrain. The two main correctness problems in class models are *consistency* [10] and *finite satisfiability* [8]. Consistency deals with necessarily empty classes and finite satisfiability with necessarily infinite classes. Detection of consistency and finite satisfiability in full class models are EXPTIME-complete problems. Correctness for multilevel models is defined by extending the class model analogy.

Consistency and Finite Satisfiability of Schemas:

Definition 2. *A schema S_i in a multilevel ontological dimension ($S_i = \langle H_i, CM_i\rangle)_{1\leq i\leq n}$ is consistent (or satisfiable) if for every class C of CM_i there is a legal Herbrand instance H_i' of CM_i that extends (includes) H_i, in which the extension of C is not empty, i.e., $C^{H'} \neq \emptyset$.*[6] *A schema S_i is finitely satisfiable if for every class C of CM_i there is a legal finite Herbrand instance H_i' of CM_i that extends (by inclusion) H_i, in which the extension of C is not empty.*

Proposition 1. *For a schema $S_i = \langle H_i, CM_i\rangle$ in a multilevel ontological dimension: S_i is consistent (respectively, finitely satisfiable) if and only if there is a legal (respectively, finite) Herbrand instance H_i' of CM_i that extends H_i, in which the extension of all classes of CM_i are not empty.*

[6] Weaker definitions are possible, following [2].

The proof is based on the closure under disjoint union of legal instances [8]. Hence, can be checked first at the class component, followed by checking the Herbrand instance component for being extendable.

Consistency and Finite Satisfiability of a Multilevel Ontological Dimension: Correctness in a multilevel dimension can be affected by inter-layer constraints. For example, in level L1 of Fig. 1, the association cycle `deskOffice, deskCompatibility, deskOS` includes *redundant* cardinalities, i.e., cardinalities that cannot be realized in any finite legal instance of the class model. In particular, for the *os* property, only cardinality 1 can be used in finite legal instances, while cardinalities 2 and 3 force instances to be infinite [25]. Therefore, constraint *L2_1*, which affects level L0 and requires existence of an offspring of `ComputerType` that is linked to two offsprings of `OSType`, cannot hold in any finite legal instance, thereby violating finite satisfiability.

We identify three possible correctness aspects in a multilevel dimension $\Theta = (S_i = \langle H_i, CM_i \rangle)_{1 \leq i \leq n}$:

1. **Weak local correctness**: Θ is *consistent* (respectively, *finitely satisfiable*) if for every class C in a schema S_i, there is a legal (respectively, finite) instance of Θ in which the extension of C is not empty.
2. **Strong local correctness**: Θ is *consistent* (respectively, *finitely satisfiable*) if for every schema S_i, there is a legal (respectively, finite) instance of Θ in which the extension of all classes of S_i are not empty.
3. **Global correctness**: Θ is *consistent* (respectively, *finitely satisfiable*) if there is a legal (respectively, finite) instance of Θ such that for every schema S_i the extension of all classes of S_i are not empty.

By Proposition 1, weak local correctness implies strong local correctness, so we will deal just with *local correctness*. However, local correctness does not imply global correctness since, as noted above, inter-level constraints may interact.

Validation and analysis of a multilevel model involves checking global and local correctness as well as instance completion, testing, and query answering. Global validation requires correctness checking techniques that operate in a multilevel domain. The FOML implementation described in Sect. 4, checks local and global validation, performs testing, and enables query answering.

4 Multilevel Modeling in FOML

FOML [18] is intended to support model-level activities, such as constraints (extending UML diagrams), dynamic compositional modeling (intensional and transformational), analysis and reasoning about models, model testing, and meta-modeling. In [6] FOML was suggested for multilevel modeling based on its uniform treatment of types and instances and of both abstract syntax and semantics. Moreover, as an executable (i.e., operational) logic language, FOML can express and reason about multiple crosscutting multilevel dimensions, including instantiation constraints. In this paper, we extend this argument by claiming that the FOML encoding is provably correct.

FOML is a semantic layer on top of a compact logic rule language of *guarded path expressions*, called *PathLP* [7,19], an adaptation of a subset of F-logic [21]. Overall, PathLP provides reasoning services over unrestricted instance-of, subtype, and object-link relations, while FOML provides the modeling framework.

4.1 PathLP

The main syntactic constructs of PathLP are *path expressions* for *objects* and *types*, *membership* and *subtyping* relations, *facts, rules, queries*, and *constraints.* **Path expressions** have the general form of `root.link[guard]`, where `root`, `link`, and `guard` are terms that denote semantic entities, and the link applied at `root` evaluates to a set that contains `guard`. For example, referring back to Fig. 1, the `os` values for the object `Precision` can be defined like this:

```
Precision.os[Linux,Windows10];
```

This expression states that `Linux` and `Windows10` belong to the set of OS's for Dell's computer model `Precision`. The target class and multiplicity of property `os` is further constrained at level L1 by a **type path expression**:

```
DesktopType!os[DeskOSType]{1..3};
```

This says that class `DesktopType` has a property `os` that yields objects that must belong to class `DeskOSType` and, for each `DesktopType` object, the property `os` has at least one and at most three objects.

PathLP includes two semantically supported special relations `::` for subtyping and `:` for membership. For example,

```
SystemType::SoftwareType;    OSType:SystemType;    DeskOSType:OSType;
```

are **facts** that declare `SystemType` as a subclass of `SoftwareType`, and `OSType` and `DeskOSType` as members of the classes `SystemType` and `OSType`, respectively. Subtyping is interpreted as a partial order, and PathLP supports the usual properties of subtyping and membership.

Links in path expressions can take arguments. This is used to account for methods. For instance, in Fig. 1, the `HardwareType` class has a method `tested`, which takes a *date*-argument and returns a string.

```
ComputerType:HardwareType;    ComputerType.tested(20170412)[failed];
```

Formally, the link `tested(p)`, when applied to the `ComputerType` object with the argument 20170412 (12th of April, 2017), yields a set that contains the symbol `failed`. This value can be further constrained to be unique using a type path expression at level L3:

```
HardwareType!tested(year)[String]{1..1};
```

PathLP uses the regular Logic Programming nomenclature of **facts, rules, queries**, and **constraints**. Rules represent implications and are recognized via the symbol `:-`, which separates the *head* (conclusion, on the left) from the *body* (premise, on the right). For example, the rule `L2_1` in Fig. 1 is represented by two PathLP rules for the classes at levels L2 and L1, as follows:

$$\begin{array}{ll} ?\texttt{cT.appl}[?\texttt{cApplT}] :- & ?\texttt{dkT.appl}[?\texttt{cApplT}] :- \\ \quad ?\texttt{cT} : \texttt{ComputerType}, & \quad ?\texttt{dkT} : \texttt{DesktopType}, \\ \quad ?\texttt{cT.os.applCompat}[?\texttt{cApplT}]; & \quad ?\texttt{dkT.os.officeCompat}[?\texttt{officeT}]; \end{array} \tag{1}$$

These rules have *variables*, which are symbols prefixed with "?". Later we show how to represent these rules with a single recursive FOML rule, using the multilevel `offSpring` operation, regardless of how many offspring classes are involved.

Queries are recognized by the prefix `?-`. They are used to retrieve information that is derivable from the specification. For example, the following query retrieves pairs `?OS, ?Office` of members of classes `DeskOSType` and `OfficeType` that are related via the property chain `deskHardw.office`:

```
?- ?OS:DeskOSType,?OS.deskHardw.office[?Office];
```

The answer to this query is: ⟨`Windows10`, `WinOffice`⟩, ⟨`Linux`, `WinOffice`⟩, ⟨`Windows10`, `KOffice`⟩, ⟨`Linux`, `KOffice`⟩. The last two answers come from rule `L2_1`, which infers the `deskOffice` link between `Precision` and `KOffice`.

Constraints have the prefix "`!-`" — they specify forbidden states. The following constraints enforce the disjointness of classes `SystemType`, `ApplicationType` in level L3, and restrict the `parentPart` association to be non-circular, i.e., a hardware type cannot be a direct or indirect part of itself.

```
!-?o:SystemType,?o:ApplicationType  !-?hT:HardwareType,?hT.closure(part)[?hT];
```

4.2 FOML

FOML extends PathLP with basic class modeling facilities, similarly to the OMG MOF. The metalevel categories of FOML are *Class, Property, Association, Attribute, Object*, and *Link*. The FOML meta-modeling tools infer the syntactic structure of the class model components, as shown in the model querying examples below. These tools include a rich, extendable library of meta-level operations. Here are some examples of intensional properties:
Property composition: *objects ?o and ?v are related via* `compose(?p1,?mid,?p2)` *if there is a path ?p1.?p2 from ?o to ?v going through ?mid:*

```
?o.compose(?p1,?mid,?p2)[?v] :- ?o.?p1[?mid].?p2[?v];
```

Transitive closure: `closure(?p)` *is a property that is the transitive closure of ?p:*

```
?o.closure(?p)[?v] :- ?o.?p[?v];
?o.closure(?p)[?v] :- ?o.?p.closure(?p)[?v];
```

Property circularity: `?p` is circular if its closure is a self link for some object `?o`:

```
?p.circular[true] :- ?o.closure(?p)[?o];
```

Using these meta-modeling facilities, FOML can inspect the syntactic structure of class models and define derived relationships.

Find pairs of properties of the same association:
```
?- ?assoc._props[?prop1,?prop2];
```
Find properties that connect a class to itself:
```
?- ?Class!?prop[?Class];
```

Correctness Summary:

Claim (Correctness of FOML encoding for class models). Let CM be a UML class model, H be a Herbrand instance of CM, and let CM^{FOML},H^{FOML} be their respective FOML encodings. Then, $H^{FOML} \models CM^{FOML}$ iff $H \models CM$, i.e., H^{FOML} is valid in CM^{FOML} iff H is a legal Herbrand instance of CM. $\square$

The proof is based on a correspondence between Herbrand models of the FOML encoding to Herbrand instances of the class model.

4.3 Multi-FOML: Multilevel Modeling in FOML

MLM requires mediation between adjacent schemas. In Multi-FOML this is done by introducing Herbrand mediators between adjacent class models. A mediator specifies a class model as a partial instance of its immediate higher class model, including mapping of names. For example, the specification of level L1 in Fig. 1, as an Herbrand instance of L2:

```
DesktopType.appl[OfficeType];     MobileType.appl[OfficeType];
DesktopType.os[DeskOSType];       OfficeType.osCompat[DeskOSType];
```

All other components of the Herbrand instance are inferred by FOML, using the meta-modeling tools.

The MLM support of multi-FOML includes computation of levels, potency and offsprings, attribute inheritance, inter-layer (deep) constraint and rule computation, local schema validation, global validation, and the regular FOML support for on the fly querying, testing and model analysis. Due to space limitations we just show a few.

Inter-layer Constraints and Rules. We show Multi-FOML representations for rule `L2_1@2` and constraint `L3_1@3` from Fig. 1.

```
?c.?can_use[?apl] :-                                        %% rule L2_1@2
  ComputerType.offspring(?N)[?c], ?N=<2,
  CompApplType.offspring(?N)[?apl], os.propspring(?N)[?comp_os],
  applCompat.propspring(?N)[?compat], ?c.?comp_os.?compat[?apl];
```

This rule replaces the two rules in (1). In general, one higher-layer rule with attached potency can replace many rules at lower layers, and this is the primary reason for this kind of abstraction in multilevel modeling.

```
!- HardwareType.offspring(?N)[?hw], ?N=<3,        %% constraint L3_1@3
   ApplicationType.offspring(?N)[?sw],
   softw.propspring(?N)[?swprop], ?hw.?swprop.[?sw],
   not (SystemType.offspring(?N)[?sw2],
        sysSoft.propspring(?N)[?sysprop]
        ?hw.?sysprop[?sw2]);
```

Here `not` is read as "not exists `?sysprop` such that ..." The FOML code is the negative form of the English description of constraint `L3_1@3` in Fig. 1.

Claim (Correctness of multi-FOML encoding for MLM). Let Θ be a multilevel dimension and Θ^{FOML} be its FOML encoding. Then Θ^{FOML} is valid iff Θ is globally correct. □

The proof is based on a correspondence of the Herbrand-based theories.

5 Related Work

Most MLM tools support modeling activities, but do not provide analysis, validation, or other reasoning services, and are not built on formal MLM theory. Semantic approaches to multilevel modeling include the graph-based theory of [28], and axiomatic approaches such as [12] (FOL) or [14] (OCL). The first approach views multilevel models as graphs that simulate the typed-by and conforms-to relations. The levels are implicitly defined by the structure of the graph. Assignment of potency to elements, and graph flattening rules define the semantics of inheritance. In the axiomatic approach, the full multilevel theory is encoded with the help of large sets of axioms. Both approaches rely on indirect translational semantics, and. it is unclear how well they integrate with existing class model analysis tools.

Telos [26], a rich knowledge representation framework implemented in ConceptBase [17], is based on first-order logic. It supports unrestricted instance-of and subtyping chains, but its reasoning capabilities and expressive power are incomparable to those of FOML (neither subsumes the other). RDF [22] is yet another language that supports unrestricted instance-of and subtyping, but its expressive power is too limited for our purposes. Nivel [3] is another logic-programming based language designed for multilevel metamodeling. In comparison, FOML focuses on software modeling, has simpler path-expression based syntax, and greater expressivity. The expressivity gap widens further when FOML is extended with Transaction Logic [11] and HiLog [13], which enable logic reasoning about statecharts and support higher-order introspection. Neither of these languages support the desiderata for multilevel modeling languages listed in [6].

Recently a number of works [16,27] have proposed to use F-logic [20] as a language for MLM. Indeed, FOML is a derivative of F-logic, but without sacrificing expressiveness, FOML is a much smaller language and its constructs are designed specifically for modeling. The aforesaid works do not provide declarative semantics for MLM, which is a main contributions of the present paper.

A comparison of FOML with Alloy and OCL appears in [5]. FOML appears to subsume and extend the *representational* capabilities of both languages. As an analysis tool, FOML provides services not supported by OCL and Alloy tools, including querying, inference, meta-level reasoning, and MLM.

6 Conclusion and Future Work

We presented a formal theory of multilevel modeling, including analysis and correctness. The theory is validated by (1) encoding it in the executable FOML language; (2) showing that it can account for all MLM features; (3) on proving the correctness of the encoding. Specifically, we showed how FOML is used for formal logic specification of class models, their instances, and constraints; querying and reasoning about multilevel models; and correctness analysis. The expressivity of FOML goes beyond UML's class models and has the necessary wherewithals to be a compact, yet expressive underlying framework for MLM. In future work, we will study the pragmatics of multilevel modeling, including methodologies for breaking monolithic class-hierarchies into multilevel structures.

References

1. Acherkan, E., Hen-Tov, A., Lorenz, D., Schachter, L.: The ink language metametamodel for adaptive object-model frameworks. In: OOPSLA 2011 (2011)
2. Artale, A., Calvanese, D., Ibáñez-García, A.: Full satisfiability of UML class diagrams. In: Parsons, J., Saeki, M., Shoval, P., Woo, C., Wand, Y. (eds.) ER 2010. LNCS, vol. 6412, pp. 317–331. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16373-9_23
3. Asikainen, T., Mannisto, T.: Nivel: a metamodelling language with a formal semantics. Softw. Syst. Model. (SoSyM) **8**(4), 521–549 (2009)
4. Atkinson, C., Kühne, T.: Rearchitecting the uml infrastructure. ACM TOMACS **12**(4), 290–321 (2002)
5. Balaban, M., Bennett, P., Doan, K.H., Georg, G., Gogolla, M., Khitron, I., Kifer, M.: A comparison of textual modeling languages: OCL, Alloy, FOML. In: 16th International Workshop on OCL and Textual Modeling, Models (2016)
6. Balaban, M., Khitron, I., Kifer, M.: Multilevel modeling and reasoning with FOML. In: IEEE CS International Conference on SwSTE 2016 (2016)
7. Balaban, M., Kifer, M.: Logic-based model-level software development with F-OML. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 517–532. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24485-8_38
8. Balaban, M., Maraee, A.: Finite satisfiability of UML class diagrams with constrained class hierarchy. ACM TOSEM **22**(3), 24:1–24:42 (2013)
9. Balaban, M., Maraee, A.: UML Class Diagram: Abstract syntax and Semantics (2017). https://goo.gl/UJzsjb
10. Berardi, D., Calvanese, D., Giacomo, D.: Reasoning on UML class diagrams. Artif. Intell. **168**, 70–118 (2005)

11. Bonner, A.J., Kifer, M.: A logic for programming database transactions. In: Chomicki, J., Saake, G. (eds.) Logics for Databases and Information Systems. SECS, vol. 436, pp. 117–166. Springer, Boston (1998). https://doi.org/10.1007/978-1-4615-5643-5_5
12. Carvalho, V.A., Almeida, J.P.A.: Toward a well-founded theory for multi-level conceptual modeling. Softw. Syst. Model. **17**, 205–231 (2018)
13. Chen, W., Kifer, M., Warren, D.: HiLog: a foundation for higher-order logic programming. J. Log. Program. **15**(3), 187–230 (1993)
14. Gogolla, M., Sedlmeier, M., Hamann, L., Hilken, F.: On metamodel superstructures employing UML generalization features. In: MULTI 2014 (2014)
15. Henderson-Sellers, B.: On the Mathematics of Modelling, Metamodelling, Ontologies and Modelling Languages. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-29825-7
16. Igamberdiev, M., Grossmann, G., Selway, M., Stumptner, M.: An integrated multi-level modeling approach for industrial-scale data interoperability. Softw. Syst. Model. **17**(1), 269–294 (2018)
17. Jarke, M., Gallersdörfer, R., Jeusfeld, M., Staudt, M., Eherer, S.: ConceptBase - a deductive object base for meta data management. J. Intell. Inf. Syst. **4**, 167–192 (1995)
18. Khitron, I., Balaban, M., Kifer, M.: The FOML Site (2017). https://goo.gl/AgxmMc
19. Khitron, I., Kifer, M., Balaban, M.: PathLP: a path-oriented logic programming language. The PathLP Web Site (2011). https://goo.gl/877S43
20. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**(4), 741–843 (1995)
21. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM **42**, 741–843 (1995)
22. Klyne, G., Carroll, J.J.: Resource description framework (RDF): concepts and abstract syntax (2006)
23. de Lara, J., Guerra, E., Cuadrado, J.: When and how to use multilevel modelling. ACM TOSEM **24**(2), 12:1–12:46 (2014)
24. de Lara, J., Guerra, E., Cuadrado, J.S.: Model-driven engineering with domain-specific meta-modelling languages. SoSyM **14**(1), 429–459 (2013)
25. Maraee, A., Balaban, M.: Removing redundancies and deducing equivalences in UML class diagrams. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 235–251. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11653-2_15
26. Mylopoulos, J., Borgida, A., Jarke, M., Koubarakis, M.: Telos: representing knowledge about information systems. ACM TOIS **8**(4), 325–362 (1990)
27. Neumayr, B., Schuetz, C.G., Jeusfeld, M.A., Schrefl, M.: Dual deep modeling: multi-level modeling with dual potencies and its formalization in F-Logic. Softw. Syst. Model. **17**(1), 1–36 (2016)
28. Rossini, A., de Lara, J., Guerra, E., Rutle, A., Lamo, Y.: A graph transformation-based semantics for deep metamodelling. In: Schürr, A., Varró, D., Varró, G. (eds.) AGTIVE 2011. LNCS, vol. 7233, pp. 19–34. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34176-2_4