Mediation-Based MLM in USE

Mira Balaban*
Computer Science Department,
Ben-Gurion University of the Negev
Israel
mira@cs.bgu.ac.il

Lars Hamann
Computer Science Department,
Hamburg University of Applied Sciences
Germany
lars.hamann@haw-hamburg.de

Gil Khais
Software Engineering Program,
Ben-Gurion University of the Negev
Israel
gil4390@gmail.com

Amiel Amram Saad
Software Engineering Program,
Ben-Gurion University of the Negev
Israel
amielsaa@post.bgu.ac.il

Azzam Maraee
Managerial Information Systems
Department, Achva Academic College
Israel
mari@cs.bgu.ac.il

Arnon Sturm
Software and Information Systems
Engineering, Ben-Gurion University
of the Negev
Israel
sturm@bgu.ac.il

Abstract

Multi Level Modeling (*MLM*) has been around in the modeling community for over twenty years. It has attracted much attention, both on theoretical and practical grounds. Multiple approaches have emerged, with different support for MLM concepts on the levels of syntax, semantics and pragmatics. MLM tools support a variety of applications, ranging from ontology specification to software modeling.

This paper introduces the MedMLM-USE tool, which implements the Mediation-based MLM theory as an extension of the USE tool. The USE tool has been selected due to its open, well-structured architecture. The MedMLM-USE application extends: (1) the well-defined meta-model that is supported by USE, with MedMLM concepts; (2) the USE modeling services to support MLM-models. This paper describes the extended meta-model and services, provides examples, defines an inheritance semantics for instance-of, and discusses engineering difficulties.

CCS Concepts

• **Computing methodologies → Modeling methodologies**.

Keywords

Multi-level modeling, MLM semantics, MLM implementation

ACM Reference Format:
Mira Balaban, Lars Hamann, Gil Khais, Amiel Amram Saad, Azzam Maraee, and Arnon Sturm. 2024. Mediation-Based MLM in USE. In *ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24), September 22–27, 2024, Linz, Austria.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3652620.3688214

*Supported in part by BSF Grant 2017742

1 Introduction

Multi Level Modeling (*MLM*) has been around in the modeling community for over twenty years [8, 9]. It has attracted much attention, both on theoretical and practical grounds [2, 15, 23, 24]. Multiple approaches have emerged, with different support for MLM concepts on the levels of syntax, semantics and pragmatics. MLM tools support a variety of applications, ranging from ontology specification to software modeling.

Mediation-based MLM (henceforth *MedMLM*), which is a formal theory of MLM modeling, was introduced in [13]. A model in MedMLM consists of an ordered set of *class models* that are interrelated by *instance-of* relationships among elements in adjacent class models (in a non-cyclic manner). The *instance-of* relationships between two levels are grouped in a *mediator*. A class-model and its mediator form a *level* in the model. A MedMLM model can be enriched by *inter-level* associations and constraints. MedMLM is unique in supporting a modular architecture of levels and mediators. MedMLM tools can be developed on top of plain class modeling applications.

In this paper we introduce the MedMLM-USE tool, that is built as an extension of the USE tool [32]. The USE tool has been selected due to its open, well-structured architecture, that enables relatively easy extension. The MedMLM-USE extension extends: (1) the well-defined meta-model that is supported by USE, with MedMLM concepts; (2) the USE modeling services to support MLM-models. This paper describes the extended meta-model and services, provides examples, defines an inheritance semantics for instance-of, and discuss engineering difficulties. A previous application of MedMLM was developed on top of the logic-based FOML tool [12, 35, 36].

Section 2 introduces the background on MLM, MedMLM, and USE. Section 3 describes the mediation-based USE application, with its two parts: Multi-USE and MedMLM-USE. Section 4 concludes the paper.

2 Background

The section starts with an introduction to MLM, which is followed by a sketch of the Mediation-based MLM theory from [13], and a short description of the USE modeling tool [32].

2.1 Multi Level Modeling

Figure 1 describes a multilevel model that describes PC-related products, in the light of general hardware-software modeling abstraction. The MLM model consists of two models: A `PC-level` model, that makes the bottom level, and a `ComputerProduct-level` above it.

Some classes and associations in the `PC-level` model are related to classes and associations, respectively, in the `ComputerProduct-level` by an *instance-of* relation, which is marked by a "**:**" operator-like notation, as in *PC:Computer*. Such classes are called *clabjects* [8, 24], i.e., ***cla****sses that are also* o***bject****s*, and such associations are called *assoclinks*, i.e., ***assoc****iations that are also* ***link****s*. In some approaches, attributes of clabjects, and roles (association-ends, properties) of assoclinks can be *renamed*. In the `PC-level` model the clabjects are *Device, PC, PCAppl, PCOS*, and the assoclinks are *pcCon, pcAppl, pcCompat, pcOs*. The roles *hardw, softw* of *hardwSoftw* are renamed in *pcOs* by *pc, os*, respectively.

An MLM model can include *inter-level associations* (*inter- association*), which are set between classes on different levels. In Figure 1, there are inter-association between `PC-level`.*PCOS* to `Computer Product-level`.*Application* and between `PC-level`.*PC* to `Computer Product-level`.*Computer*. The inter-level associations are marked by thicker lines. Inter-associations carry multiplicity constraints, as usual.

Constraints in MLM models have appeared either as *intra-constraints*, that restrict legal object-instances (states) of the class model in a single level, or *deep constraints* (*inter-constraints*) that constrain legal object instances of multiple levels. Quite frequently, deep constraints are specified as related to some level, and stated as affecting multiple levels downwards. Deep constraints might be specified as *inherited* via *offspring relationships*, i.e., the transitive closure of the *generalization* and *instance-of* relations. The exact semantics of deep constraints vary among MLM approaches.

Figure 1 demonstrates the different kinds of constraints:

- **`Non-EmptyRelatedStandard`**: *The relatedStandard attribute of a Computer object must be specified.* An intra-constraint on the `ComputerProduct-level`.
```
Constraint NonEmptyRelatedStandard
Context Computer
inv: relatedStandard !=NULL
```
- **`HardwareDates:`** The *testDate* of a *parent Hardware* object is later than those of its *part*s. Intended as an inherited deep constraint (marked with dashed borderline):
```
Constraint HardwareDates
Context Hardware
inv: part.selfDates ->
        forAll(t|t<self.testDate)
```
- **`ApplSysCompat:`** *An application that runs on a computer must have a compatible system that also runs on that computer.* Also, an inherited deep constraint.
```
Constraint ApplSysCompat
Context Application
inv: self.syst.hardw ->
        includesAll(self.hardw)
```
- **`PCInstallation:`** *An installer application for a PC operating system must run on a computer that can maintain the PC on which the operating system runs.* An inter-constraint, that involves elements from multiple level (marked with thick borderline):
```
Constraint PCInstallation
Context Application
inv: self.syst.hardw ->
       select(h | h.oclisTypeOf(Computer)).
           oclAsType(Computer).maintained)->
              includesAll(self.installed.pc)
```

Multilevel modeling is an umbrella for modeling approaches that emerged from different directions, with a variety of motivations and goals, and yielded different frameworks and systems. There are ontology-motivated approaches, logic-based knowledge representation systems, and software-modeling frameworks.

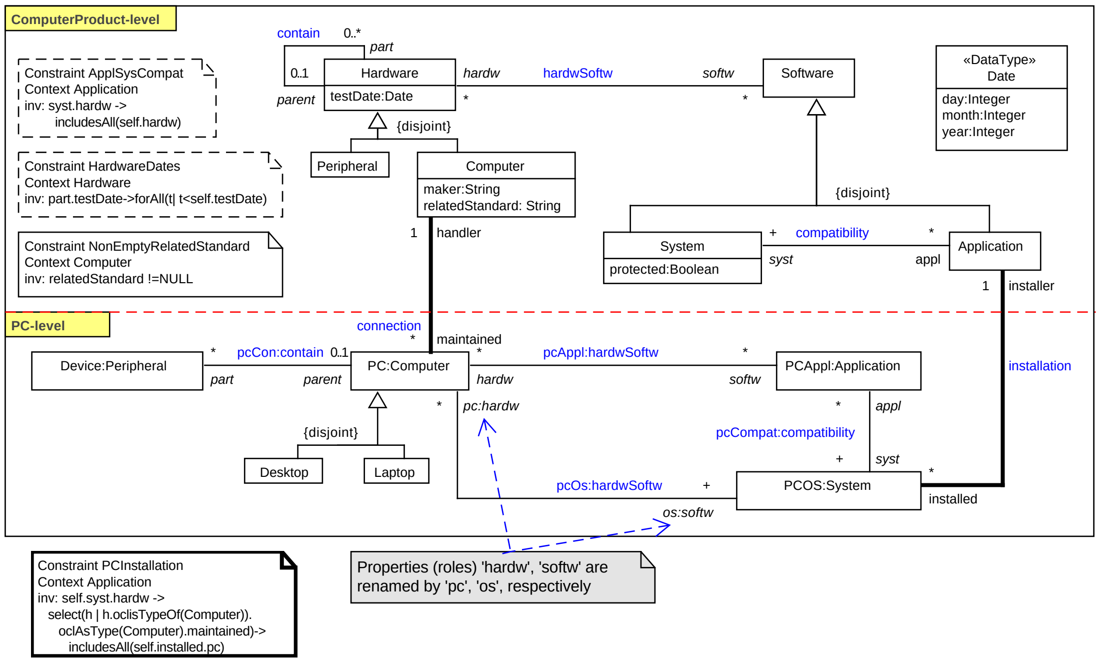
The most popular approach seems to be *potency-based MLM* [8], also termed *Deep modeling Languages* (*DML*s), which is characterized by a mechanism that assigns a *potency* numerical value to elements of the model. Potency can be assigned to a variety of elements, such as classes, associations, and attributes (we are not clear about operations).

The potency mechanism specifies modes of instantiation and inheritance of features along offspring chains of a class, i.e., classes that are related to it by the transitive closure of the *subclass* and the *instance-of* relations [7, 8, 22]. It is supported by tools like Melanie [40], MetaDepth [22] that introduces *leap potency*, and FMMLx [27]. Melanie and MetaDepth are compared in [30]. Different variants of the potency mechanism and other cross level deep relationships have been discussed in [4, 6, 22, 39–41].

The logic trend in MLM is probably rooted in the Telos [45] and F-Logic [37] knowledge representation languages that support unrestricted instance-of and subtyping chains, and therefore are natural platforms for implementing MLM. Telos yielded ConceptBase, DeepTelos [34, 47] that supports multilevel reasoning. F-logic yielded FOML [11, 12, 35], the mediation-based MLM theory (MedMLM) [13, 16] that is described below, and several industrial applications [44, 48] that are implemented in the Flora [25] application of F-logic. Nivel [3] is another logic-programming based language designed for MLM. A different approach that relies on logic axiomatization was initiated by the *MLT*-ML2* framework that defines a semantic world with pre-defined global organization of *orders* and *categorization* [1, 21, 26].

Other forms of inter-level relationships, that possibly enable further modeling flexibility include *concretization*, *categorisation, extension* and *refinement* meta-relationships [46, 49]. Different forms of cross-level deep modeling and feature inheritance are supported by in [21, 28, 29, 43, 48]. Several comparisons of MLM approaches and tools have appeared in [5, 10, 14, 26, 28, 33, 44, 50].

Most MLM frameworks include some form of explicit syntactic specification of the levels of elements [14]. Sometimes, levels of elements are used for the syntactic organization of elements into distinct namespaces, as in a *packaging mechanism.* In such frameworks, levels might carry significant modeling role, such as instantiation restrictions, and inter-level, and intra-level meta-modeling rules [22, 33, 40]. In the MedMLM theory, the *level* concept is a basic syntactic and semantic building block.



```
Constraint ApplSysCompat
Context Application
inv: syst.hardw ->
       includesAll(self.hardw)
```

```
Constraint HardwareDates
Context Hardware
inv: part.testDate->forAll(t| t<self.testDate)
```

```
Constraint NonEmptyRelatedStandard
Context Computer
inv: relatedStandard !=NULL
```

```
Constraint PCInstallation
Context Application
inv: self.syst.hardw ->
   select(h | h.oclisTypeOf(Computer)).
     oclAsType(Computer).maintained)->
       includesAll(self.installed.pc)
```

Thick lines denote inter-associations. Constraints that are specified within dashed borders indicate deep constraints. Multiple level constraints are written outside the levels drawing.

Figure 1: A multilevel model

2.1.1 *Instance-of semantics in MLM.*

In [17] we raised the issue of type safety in MLM-based software that enables selected inheritance. The problem is that most MLM approaches assume *instance-of* (and other inter-level meta-relationships) semantics that enables selected inheritance of features along offspring chains. Selected inheritance implies a violation of the main characteristic of inheritance: Objects of a subtype preserve all properties of objects of its supertypes (Liskov [42]). This characterization enables the essential client visibility feature: Client functions of a type hierarchy are oblivious to the exact type identity of their arguments[1].

There are three options for *instance-of* semantics in an MLM-based software: (1) No feature inheritance; (2) Full feature inheritance; (3) Selected feature inheritance. The first option adopts the semantics of set membership: As an object of its class, it populates the features of the class, but these features do not characterize its class view[2]. That means that attributes and (roles in) associations of a class refer only to its objects, and do not apply to objects and assoclinks of its clabjects and assoclinks, respectively. For example, in Figure 1, objects of class *Computer* have a *maker*, but objects of *PC* do not have a *maker* attribute.

The second option means that *instance-of* semantics is identical to subtyping. So, simply – why needed? The third option means that client visibility over offspring chains cannot be taken for granted. There is a need to develop a feasible typing theory that would enable selected feature inheritance that guarantees type safety. In [17] we suggested a mechanism that might achieve this goal.

2.2 The Mediation-based MLM Theory

The mediation-based MLM theory is built on top of the standard set-theoretic formulation of the *Class Model* (see, for example, the Class Model definition in [12, 19]), using *mediators* for relating multiple levels of class models.

A *MedMLM model* consists of a totally ordered set of inter-related *levels*. A level consists of a *class model* and an interlevel *mediator*[3]. The class model of a level is termed the *class-view* (*class-facet*) of the level. The mediator of a level with class model *CM*, specifies an *object-view* (*object-facet*) of *CM* as an *object instance* of the class model in the immediate higher level. The mediator of the top level specifies an empty object view (empty object instance). Figure 2 shows the MedMLM object-view of the PC-level in the MLM model in Figure 1.

A MedMLM model is *well-defined* if: (1) the levels are totally ordered; (2) the object view of every level (apart from the top one) is a *partial instance* of the next higher level. The MLM model in Figure 1 is well defined since the object-view of the bottom level is

[1]Note that the *redefines* UML constraint which can be defined between roles of a subclass and its superclass, also violates these characteristics.
[2]Indeed, in Flora [25], inherited attributes are marked as static, i.e. not inherited by members of members.

[3]In principle, there is also the notion of *MLM dimension*, where a MLM model consists of multiple, possibly inter-related, modeling *dimensions*, where each dimension is constructed from levels. However, in practice, the multiple dimensions option is not really used. Therefore we ignore it.

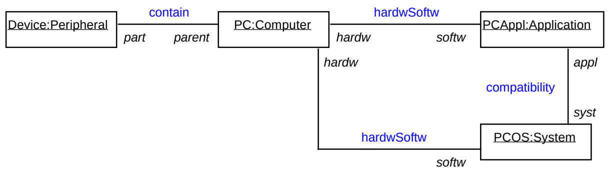


Figure 2: The object view of `PC-level` in Figure 1

a legal instance of the top level. The well-definedness restriction is motivated by the understanding that the levels of a MedMLM model must demonstrate an essential inter-dependency. That is, the levels cannot be just any arbitrary collection of class models.

The theory is inspired by Herbrand semantics, which enables syntactic management of semantic models. In MedMLM, syntactic correctness (well-definedness) is defined by creating a syntactical object-view, and checking it semantically, as a partial instance of the next higher level.

The semantics of a MedMLM model is defined by its legal object instances. An *instance of a MedMLM model* consists of instances of the level class models, possibly inter-linked by the inter-associations. A legal instance of a MedMLM model consists of *legal instances of the class models* of the levels, that satisfy all inter-association multiplicities and all inter-constraints. In practice, the intended instances of a MedMLM model consist of legal instances of the bottom level, that might be extended with inter-linked objects of other levels, following multiplicity constraints on inter-associations, and inter-constraints.

The MedMLM theory does not assign semantics to the *instance-of* inter-level relationships. This is left open to MedMLM applications. The logic-based FOML application [13, 35, 36] enables attribute inheritance along instance-of relationships, using potency assignments. The MedMLM-USE application that is described in this paper, supports default inheritance of attributes and roles, with optional banning and renaming.

Mediation-based MLM is the only MLM semantic theory in which the *level* notion has a *first-class citizen* status. The class view of a level is an independent component that can be developed and maintained independently. The modular component-wise structure turns MedMLM models easy to manage, reuse, and implement on top of standard class modeling tools.

2.3 The USE Modeling Tool

The UML-based Specification Environment (USE) is a robust software modeling tool designed to support the specification, validation, and verification of UML (Unified Modeling Language) models [32]. It provides a comprehensive environment for analyzing UML models, specifically focusing on class diagrams and OCL (Object Constraint Language) constraints. USE is particularly valued for its ability to help developers ensure the correctness and consistency of (semi-)formal models through rigorous validation and, to a certain extent, verification capabilities.

2.3.1 *Architecture of USE.*

The architecture of USE is modular, comprising several key components that work together to provide a seamless modeling and verification experience. These components include:

- **Parser and Interpreter:** USE includes a robust parser and interpreter for OCL. The parser checks the syntax of OCL expressions, while the interpreter evaluates these expressions within the context of the instantiated UML model. This ensures that constraints are syntactically correctly defined and consistently applied.
- **Simulator:** The simulator enables users to create and manipulate instances of the model elements, allowing for dynamic interaction with the model. Users can instantiate classes, set attribute values, and establish links between instances to simulate real-world scenarios. Consistency of the model against the defined OCL constraints can be checked, too. This helps identify violations and inconsistencies, providing detailed feedback to the user.
- **Model Validator:** USE includes a model validator[31, 38] that performs in-depth analysis of models. It can check for properties such as model completeness, redundancy, and potential conflicts. Internally, the model validator applies SAT-solver to perform these checks. Technically, the validator is developed as a plugin.
- **User Interface:** The user interface is built on top of the modeling logic to allow for using USE as a library. The user interface itself is designed to be intuitive and user-friendly, providing easy access to all the features and functionalities of USE.

2.3.2 *Use Cases in Verifying Formal Models.*

USE is widely used in various scenarios to verify the correctness and validity of formal models. Some of the primary use cases include:

- **Consistency Checking:** USE helps ensure that UML models are internally consistent and free from contradictions. By validating OCL constraints, developers can detect and resolve issues early in the modeling process, reducing the risk of errors in later stages of development.
- **Model Validation:** Developers apply USE to validate models against specific requirements and use cases. By simulating different scenarios and checking the satisfaction of OCL constraints, USE provides a powerful means to verify that models meet the intended specifications.
- **Behavior Simulation:** The simulator feature allows users to create dynamic instances of the model and explore their interactions. This is particularly useful for testing how the model behaves under various conditions and ensuring that it performs as expected.
- **Education and Training:** USE is also used in academic settings to teach UML modeling and formal methods. The possibility to instantiate models and to evaluate expressions at runtime provides an excellent platform for students to learn and practice model specification and verification. For

instance, the authors of [20] *"[..] discovered that students appreciate very much the possibility of running their models, simulating their behavior."*

In summary, the UML-based Specification Environment (USE) is a versatile and powerful tool that plays a crucial role in the specification, validation, and verification of UML models. Its modular architecture and extensive features make it an invaluable asset for developers and educators in ensuring the correctness and consistency of formal models.

3 Mediation-Based MLM in USE

As described in the last section, USE architecture is centered around a well-defined meta-model of software models. The architecture is open, and enables extension without harming the current system. The *USE-based MedMLM* (*MedMLM-USE*) application is built around this feature. It smoothly extends the meta-model, to support MedMLM models, and implements MedMLM services on top. Naturally, there are engineering problems that must be solved, e.g., how to integrate the mediation-based MLM into USE. Two possibilities for this question were discussed:

(1) Providing a Med-MLM plugin
(2) Extending the USE project by creating a Med-MLM fork of USE

Unfortunately, option one could not be chosen, because of lacking capabilities to extend the internal USE meta-model with new elements. Here, the plugin infrastructure of USE misses useful extension points for plugins. Otherwise, this option would be the first option to choose because it does not change the stable core of USE.

Option two allows for a nearly unrestricted modification of USE. However, changes to the core were kept to a minimum, to be able to easily update the downstream project MedMLM-USE and to be able to still use the Model Validator plugin without modifications[4].

In this section, we describe how MedMLM extends the USE meta-model, introduce and set examples for the services that *MedMLM-USE* provides, and elaborate on some of the more subtle issues that have different, sometimes contradicting, aspects.

Overall, MedMLM-USE provides standard modeling services, as in USE: (1) Definition of MedMLM models, with inter-associations and inter-constraints. The model-input service also checks the model legality, i.e., syntactic correctness; (2) Consistency validation of a MedMLM model; (3) Instance checking; (4) Some help with visualization of MedMLM models.

The MedMLM is developed in two stages by extending the USE project[5]. In the first stage, USE is extended to support *Multi-models*, i.e., models that are composed of possibly inter-related internal models. The inter-relations include associations between classes in different internal models, association class constraints on such associations, and inter-constraints, i.e., constraints that involve elements in different internal models. In the second stage, the *MedMLM* model is introduced, as a special case of Multi-models.

[4]This topic needs to be further investigated, because it would allow for a rigorous verification of MedMLM-USE models. However, small changes might be required to translate MedMLM-USE models into plain USE models
[5]https://github.com/Gil4390/useBGU

3.1 *Multi-USE*: The Multi-model Extension of USE

A multi-model is a mega-model that consists of multiple, possibly inter-related and inter-constrained models. The USE extension is based on a smooth extension of the meta-model that USE devised for its implementation, and extension of USE services. Figure 3 shows a Multi-model version of the MLM model in Figure 1 (i.e., without considering instance-of links and their implications). The multi-model consists of two class models and includes the inter-associations, and all constraints (intra- and inter-), but does not have the inter-level *instance-of* relationships. Inter-constraints are not part of any component.

3.1.1 Multi-model Extension of the USE Meta-Model and Services.

Figure 4 shows the extension of the meta-model that USE supports, with Multi and MultiLevel modeling in the MedMLM approach. The extensions are emphasized using different colors. The meta-model enables a simple class model – meta-class MInternalModel, or a Multi-model that aggregates simple class models – meta-class MMultiModel. MMultiModel is a subclass of MModel, implying that it inherits all MModel features. The main modeling services are as follows:

Multi-model definition: The multi-model extension supports text input. The textual definition consists of the specification of the internal models, which is followed by the inter-associations and inter-constraints. A visual view is enabled using the USE model visualization, extended with component-model coloring. Listing 1 includes the textual specification of the multi model in Figure 3.

Semantics – model validation: An *instance of a multi-model* consists of instances of its internal models, with objects that are possibly linked by inter-associations. A *legal instance* is one that satisfies all constraints (including the inter-constraints). A multi-model is *consistent* (*satisfiable*) if it has a non-empty legal instance.
The multi-model extension of USE relies on the USE model validation service, for checking satisfiability of a multi-model. For example, the Multi-model in Figure 3 is consistent[6]. The engineering issues that were encountered in the implementation of this service are summarized in the next subsection.

Check instance: Instance checking in Multi-USE smoothly extends instance checking in USE. Moreover, instance checking can distinguish *partial instance*, i.e., an instance that is illegal but can be completed into a legal one by adding objects or links. Partial instances differ from illegal instances that violate constraints that cannot be repaired by the addition of objects or links. In Multi-USE, instances are introduced as a soil text file of USE.
Figure 5 shows a partial instance of the Multi-model in Figure 3. Multi-USE detects that this is a partial instance, since minimum multiplicity constraints are violated: Object `win`

[6]This multi-model would have been inconsistent if it is extended with: (1) a new inter-association between *Hardware* and *PC*, with multiplicities 1 . . 1 on both roles; (2) multiplicity constraint on the *maintained* role is modified into to 1 [18].

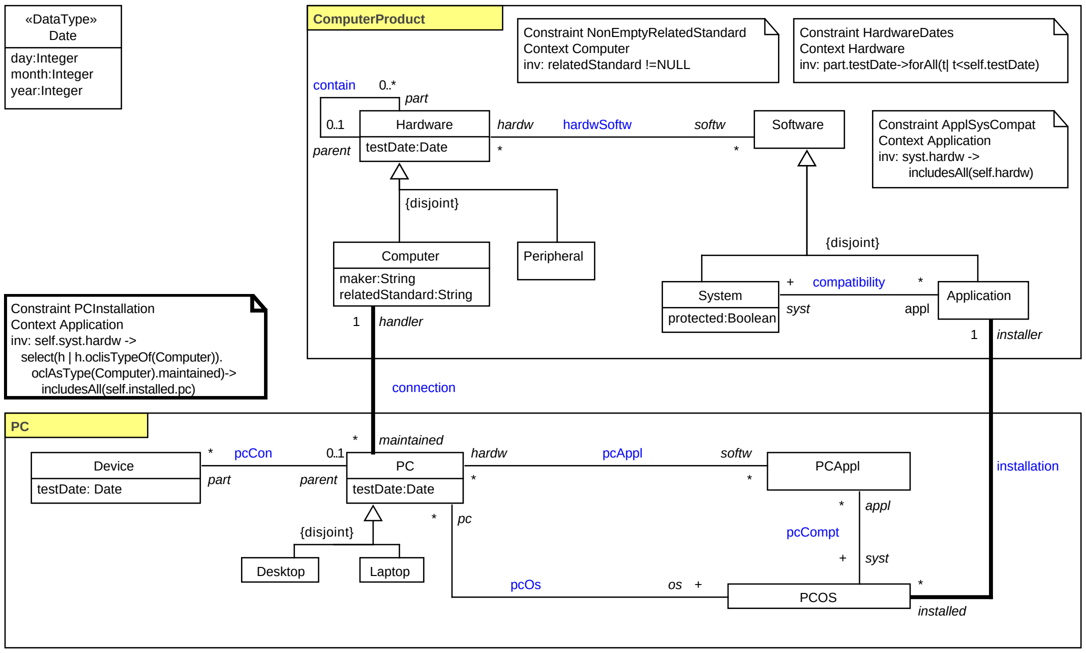
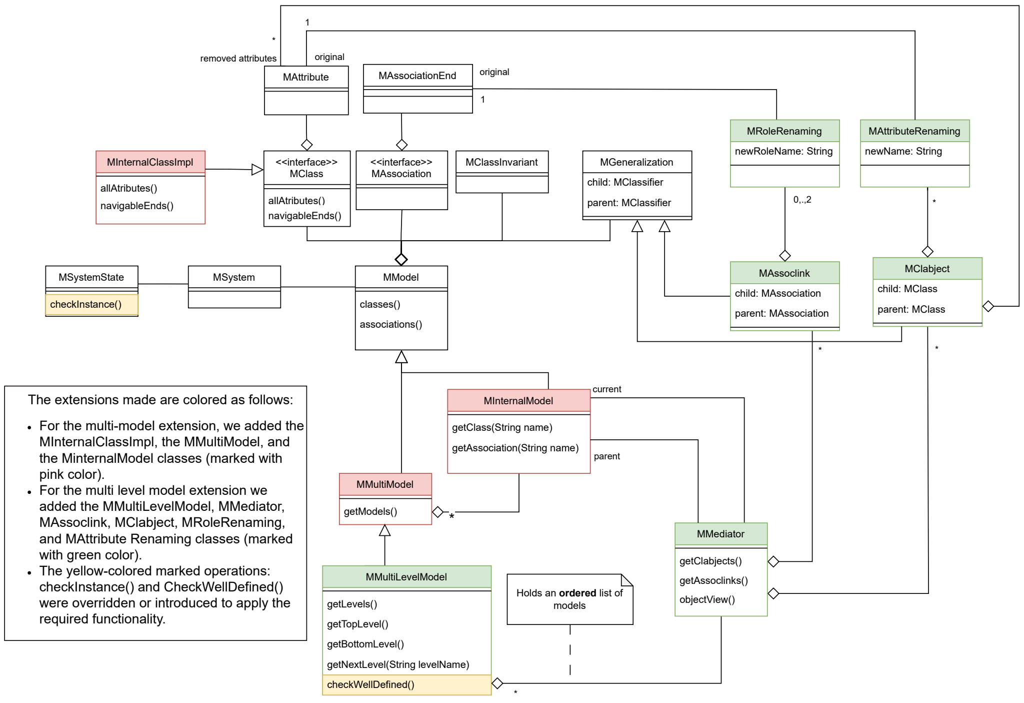


Figure 3: A multi model version of the MLM model in Figure 1



The extensions made are colored as follows:

- For the multi-model extension, we added the MInternalClassImpl, the MMultiModel, and the MinternalModel classes (marked with pink color).
- For the multi level model extension we added the MMultiLevelModel, MMediator, MAssoclink, MClabject, MRoleRenaming, and MAttribute Renaming classes (marked with green color).
- The yellow-colored marked operations: checkInstance() and CheckWellDefined() were overridden or introduced to apply the required functionality.

Figure 4: The meta-model of the *Multi-USE and MedMLM-USE extension of USE*

Listing 1: Textual input of the Multi-model in Figure 3

```
multi_model ABCD

  model Computer_product
    class Hardware
      attributes
        testDate: Integer
    end
    class Computer < Hardware
      attributes
        maker: String
    end
    class Peripheral < Hardware
    end
    class Software
    end
    class System < Software
      attributes
        protected: Boolean
    end
    class Application < Software
    end

    association hardwSoftw between
      Hardware[0..*] role hardw
      Software[0..*] role softw
    end
    association compatibility between
      System[1..*] role syst
      Application[0..*] role appl
    end
    association contain between
      Hardware[0..*] role part
      Hardware[0..1] role parent
    end

    constraints
    context Application
     inv ApplSysCompat:
       self.syst.hardw-> includesAll(self.hardw)

     context Hardware
     inv: part.testDate->forAll(t| t<self.testDate)
```

```
model PC
  class Device
    attributes
      testDate: Integer
  end
  class PC
    attributes
      testDate: Integer
  end
  class Desktop < PC
  end
  class Laptop < PC
  end
  class PCAppl
  end
  class PCOS
    attributes
      openSource: Boolean
  end

  association pcCon between
    Device[0..*] role part
    PC[0..1] role parent
  end
  association pcAppl between
    PC[0..*] role hardw
    PCAppl[0..*] role softw
  end
  association pcCompat between
    PCAppl[0..*] role appl
    PCOS[1..*] role syst
  end
  association pcOs between
    PC[0..*] role pc
    PCOS[1..*] role os
  end
```

```
  inter-associations
   association connection between
     Computer_product@Computer[1] role handler
     PC@PC[0..*] role maintained
   end
  association installation between
   Computer_product@Application[1] role installer
     PC@PCOS[0..*] role installed
  end

  inter-constraints
   context Computer_product@Application inv PCInstallation:
         syst.hardw->  select(h | h.oclIsTypeOf(Computer_product@Computer))->
         collect(c|c.oclAsType(Computer_product@Computer).maintained)->
            includesAll(self.installed.pc)
```

of *PCOS* needs an *installer Application*, and object `cloudPC2` of *PC* must have a *handler Computer*. If missing links are added, then that will be a legal instance of Figure 3. Note also that the constraints ApplSysCompat and HardwareDates, do not apply in component PC.

3.1.2 Engineering Issues in the Multi-model Extension of USE.

The main engineering problem at that stage concerned name conflicts among the models that are aggregated in the multi-model. Models are introduced as independent units, including their own OCL constraints. Therefore, they must keep their internal names. But, in order to exploit USE services, we need to submit USE a single standard class model. In order to avoid the heavy task of dynamic translation, Multi-USE keeps multi-models as a single class model, employing standard name prefixing.

Due to this engineering decision, Multi-USE can support also inter-model classes, i.e., classes that are not part of any component model, and also support association-class constraints on inter-associations.

3.2 *MedMLM-USE*: The MedMLM Extension of Multi-USE

A MedMLM model is a Multi-model in which the component models are totally ordered, and the elements in adjacent component-models are inter-related by *instance-of* relationships. MedMLM-USE smoothly extends the Multi-USE meta-model. The services are extended to support the leveled structure and implied element access. Figure 1 shows an MLM model with two components (levels). This model extends the Multi-model of Figure 3 with *instance-of* relationships between classes of the `PC-level` component and classes of the `ComputerProduct-level` component.

In MedMLM, the *instance-of* relationships between levels are specified in *mediator*s of levels. MedMLM-USE supports default inheritance of attributes and roles along *instance-of* relationships. Inheritance along *instance-of* can be associated with optional attribute and role renamings, and banning of attributes and roles. All are specified in the level mediators. *Instance-of* inheritance semantics affects MedMLM model validation and instance checking services, as will be explained below.

3.2.1 MedMLM Extension of the Multi-USE Meta-Model.

The MedMLM meta-model extends the Multi-USE meta-model with classes and associations that capture the inter-level interaction. MMultiLevelModel is a subclass of MMultiModel. As such, it inherits the main features of validation and checking instances. The inter-level structure is captured by the MMediator, MClabject, and MAssoclink classes. In order to support standard services of USE that depend on inheritance of features, MClabject, and MAssoclink extend the capabilities of MGeneralization, so to account also for feature inheritance that result from *instance-of* relationships. MMultiLevelModel is responsible for the total and legal ordering of the components, MMediator is responsible for the clabjects and the assoclinks (including their associated attribute and role renaming or banning), and computing the object view of the MInternalModel.

3.2.2 Semantics of Inheritance Along Instance-of Relationships in MedMLM .

MedMLM theory does not assign semantics to the *instance-of* inter-level relationships. The basic semantics is that of set membership, but that semantics refers only to the features of clabjects and assoclinks as objects and links of their *instance-of* owners – their *object view*, and does not say anything about their features as classes and associations – their *class view*.
Instance-of inheritance rules implemented in MedMLM-USE:

(1) Class C *instance-of* class D: denoted $C : D$
  (a) Default inheritance of all attributes and roles (navigable ends, properties) of D, unless,
  (b) Attribute a or role r of D is explicitly renamed into a' or r', respectively: denoted $a \longrightarrow a', r \longrightarrow r'$
  (c) Inheritance of attribute a or role r of D is explicitly banned: denoted $\sim a, \sim r$
(2) Association a *instance-of* association b, denoted $a : b$: The semantics of association *instance-of* in MedMLM-USE dictates replacement of association b by the declared association (assoclink) a. This has impact on the inheritance of roles of b, and is discussed below. This semantics of association *instance-of* turns out equal to the UML *redefinition* constraint. Roles of a can rename those of b.

The MedMLM *instance-of* inheritance policy can create seemingly contradictory specifications, e.g., "$C : D$, with $\sim r$, while

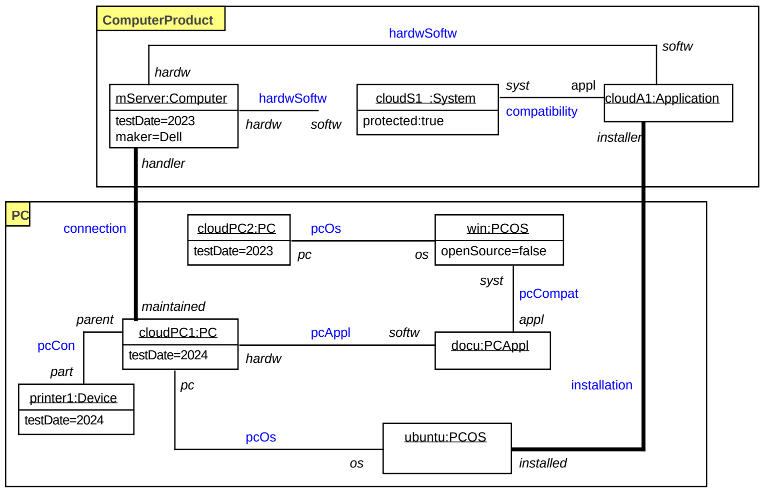


Figure 5: A partial instance of the *multi model* in Figure 3

$a : b$, where r is a role (navigable end) of D through association b", or "$C : D$, with $r \longrightarrow r'$, while $a : b$, $r \longrightarrow r''$, where r is a role (navigable end) of D through association b". At the current stage, MedMLM-USE does not check such situations. Responsibility for avoiding such specifications is set on the modeler.

3.2.3 *MedMLM Extension of Multi-USE services.*

MMultiLevelModel is a subclass of MMultiModel,and supports all services of MMultiModel: model definition, model validation, and instance checking.

MedMLM-model definition: The input definition of a MedMLM model extends the Multi-model definition with level mediators, and checking if it is *well-defined*, i.e., *syntactically correct*: (1) total ordering of the levels; (2) the object view of every level is a partial instance of its immediate higher level (see Subsection 2.2).
Listing 2 shows the textual specification of the mediators of the two levels of the MedMLM in Figure 1.

The mediators specify the *instance-of* relationships, and the inheritance of the attributes, roles. In Clabject *PC*, which is an *instance of Computer*, attribute *relatedStandard* of Computer is not inherited by *PC*, and role *softw* of association *hardwSoftw* is renamed to *os*. In clabject *PCOS*, attribute *protected* of *System* is renamed to *openSource*, while clabject *PCAppl* inherits all attributes and roles of *Application*.
As for the assoclinks, the assoclink *pcCon* is consistent with the role inheritance in clabjects *Device* and *PC*. Similarly, in assoclink *pcOS*, role renaming is consistent with role inheritance in clabjects *PC* and *PCOS*. If role renaming or banning between a clabject to its related assoclinks is contradictory, MedMLM-USE does not guarantee a definite answer. This is handled as an implementation-dependent result, and it is recommended to avoid – responsibility is set on the modeler.

Checking well-definedness: Input definition is followed by: (1) checking that the levels are totally ordered; (2) MedMLM computes the object view of every level, and applies instance checking with respect to the adjacent from the above class model (meta-class MInternalModel).

Semantics – model validation: An instance of a MedMLM model consists of instances of the level class models, possibly inter-linked by the inter-associations. An instance is legal if all level instances are legal. As explained in Subsection 2.2, in practice, intended instances consist of a non-empty instance of the bottom level, which is extended so to satisfy interlevel associations and constraints.

Listing 2: A textual specification of the mediators in the MedMLM model in Figure 1

```
mediator Computer_product < NONE
end

mediator PC < Computer_product
  clabject Device : Peripheral
  end

  clabject PC : Computer
    attributes
      ~relatedStandard
    roles
      softw -> os
  end

  clabject PCOS : System
    attributes
      protected -> openSource
    roles
      hardw -> pc
  end

  clabject PCAppl : Application
  end

  assoclink pcCon : contain
      parent->parent
      part->part
  end

  assoclink pcAppl : hardwSoftw
      softw->softw
      hardw->hardw
  end

  assoclink pcOs : hardwSoftw
      os->softw
      pc->hardw
  end

  assoclink pcCompat : compatibility
      appl->appl
      syst->syst
  end
end
```

Model validation in MedMLM-USE is similar to that of Multi-models. However, both the MedMLM-model validation and the check-instance services are affected by the inheritance semantics of *instance-of* relationships, since objects inherit attributes and roles via *instance-of* relationships.

Check instance: An intended legal instance of a MedMLM model can be built by starting with a legal instance of the bottom level, and extending it with necessary links, following inter-associations and inter-constraints.
Figure 6 presents an illegal instance of the MedMLM-model in Figure 1, with the mediator in Listing 2. The instance is illegal since the deep constraint HardwareDates is violated on object `printer1`. The constraint is inherited from class *Hardware*, via *offspring* relationships (transitive closure of generalization and *instance-of*). While in the Multi-model instance in Figure 5, objects of classes in component `PC` can violate constraints set in the `ComputerProduct` component, in the MedMLM instance deep constraints are inherited. Therefore, the two deep constraints apply on `PC-level`.
The MedMLM instance shows the effect of attribute and role inheritance policy on objects and links. In the MedMLM instance, Object `cloudPC1` has an inherited attribute *maker*, and `ubuntu` has the inherited under renaming attribute *openSource*. The *Device* object `printer1` could have been linked via role *softw* to any object of an offspring class of *Software*. In this MedMLM model, some inherited roles via *instance-of* are overridden by assoclinks that are declared in `PC-level`.

3.2.4 Engineering Issues in the MedMLM Extension of Multi-USE.

A major issue in the implementation of MedMLM-USE involves the inheritance semantics along *instance-of* relationships. It affects the validation and check-instance services, since objects in lower levels inherit features of their *instance-of* ancestors. The engineering approach taken in the implementation of *instance-of* semantics was to extend the inheritance tree of USE with a new kind of connecting edges for *instance-of.* The result is an inheritance tree that covers all *offspring* branches in the model (the offspring relation is the transitive closure of the union of the *generalization* and the *instance-of* relations). The nodes of the generalization tree span all classes of the MedMLM model, and are connected by two kinds of edges: the regular USE generalization edges, and the new MedMLM *instance-of* edges. The overall attributes and roles in a class are computed by the *allAttributes*() and *navigableEnds*() of class *MInternalImpl*.

The computation of *allAttributes*() and *navigableEnds*() for a class is activated when a MedMLM model is defined. If the inheritance procedure yields a contradictory set of attributes or roles for a class, the MedMLM model is rejected. A contradictory set is a set that forms an illegal set of attributes, e.g., a set with non-unique attribute names, or role names.

Role inheritance might create issues of type inconsistency or roles (navigable ends in the USE terminology) that are not associated with any association. For example, the mediator in Listing 2 states that clabject *PCAppl* inherits all roles of *Application*, and in particular, it inherits role *hardw*, whose type is *Hardware*, while assoclink *pcAppl* already has role *hardw* with type *PC*. Solutions to such issues are still under discussion.

4 Conclusions and Future Research

In this paper we have introduced the Mediation-Based MLM extension of the USE application. MedMLM is a formal MLM theory that can be implemented on top of a standard class modeling tool. MedMLM-USE demonstrates a smooth extension of USE, which was possible due to its well-structured architecture.

The paper includes a summary of the MedMLM theory, the USE application, and described the essence of the extension. The metamodel underlying the extension is presented, and capabilities of the extension are explained via examples. The inheritance semantics of *instance-of* in MedMLM-USE is defined and demonstrated through the examples. Finally, central, yet-to-be-solved issues in theory and practice are discussed.

MedMLM-USE is an ongoing project. Future plans include further investigation of semantic issues in MedMLM-USE, and development of implemented efficient support.

References

[1] Almeida, J., Fonseca, C., and Carvalho, V. A comprehensive formal theory for multi-level conceptual modeling. In *ER* (2017), pp. 280–294.
[2] Almeida, J. P. A., Frank, U., and Kühne, T. Multi-Level Modelling (Dagstuhl Seminar 17492). *Dagstuhl Reports 7*, 12 (2018), 18–49.
[3] Asikainen, T., and Mannisto, T. Nivel: a metamodelling language with a formal semantics. *Software and System Modeling (SoSyM) 8*, 4 (2009), 521–549.
[4] Atkinson, C., and Gerbig, R. Melanie: multi-level modeling and ontology engineering environment. In *Int. Master Class on MDE: Modeling Wizards* (2012).
[5] Atkinson, C., Gerbig, R., and Kuehne, T. Comparing Multi-Level Modeling Approaches. In *1st International Workshop on Multi-Level Modeling (Multi 2014)* (2014).
[6] Atkinson, C., Gerbig, R., and Kuhne, T. Opportunities and Challenges for Deep Constraint Languages. In *15th International Workshop on OCL and Textual Modeling* (2015).
[7] Atkinson, C., and Kühne, T. The essence of multilevel metamodeling. In *UML*. Springer, 2001, pp. 19–33.
[8] Atkinson, C., and Kühne, T. Rearchitecting the uml infrastructure. *ACM Trans. Model. Comput. Simul. 12*, 4 (2002), 290–321.
[9] Atkinson, C., and Kühne, T. Reducing accidental complexity in domain models. *Software & Systems Modeling 7*, 3 (2008), 345–359.
[10] Atkinson, C., and Kühne, T. On evaluating multi-level modeling.
[11] Balaban, M., Bennett, P., Doan, K. H., Georg, G., Gogolla, M., Khitron, I., and Kifer, M. A Comparison of Textual Modeling Languages: OCL, Alloy, FOML. In *16th International Workshop on OCL and Textual Modeling, Models* (2016).
[12] Balaban, M., Khitron, I., and Kifer, M. Logic-based software modeling with FOML. *The Journal of Object Technology 19*, 3 (2020), 3:1–21.
[13] Balaban, M., Khitron, I., Kifer, M., and Maraee, A. Formal executable theory of multilevel modeling. In *CAISE* (2018), pp. 391–406.
[14] Balaban, M., Khitron, I., Kifer, M., and Maraee, A. Multilevel modeling: what's in a level? a position paper. In *MODELS Workshops, MULTI-2018* (2018).
[15] Balaban, M., Khitron, I., and Maraee, A. Accidental Complexity in Multilevel Modeling Revisited. *Software and Systems Modeling (SoSyM) (accepted for publication)* (2021).
[16] Balaban, M., Khitron, I., Maraee, A., and Kifer, M. Mediation-based MLM in FOModeLer. In *MULTI-22, ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (2022), pp. 444–452.
[17] Balaban, M., Kifer, M., and Maraee, A. Clabject Typing in MLM – the Double Life of a Clabject: A Position Paper. In *MULTI-23, ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS-C)* (2023), pp. 635–638.
[18] Balaban, M., and Maraee, A. Finite Satisfiability of UML Class Diagrams with Constrained Class Hierarchy. *ACM TOSEM 22*, 3 (2013), 24:1–24:42.
[19] Balaban, M., and Maraee, A. UML Class Diagram: Abstract syntax and Semantics. https://goo.gl/UJzsjb, 2017.
[20] Burgueño, L., Vallecillo, A., and Gogolla, M. Teaching UML and OCL models and their validation to software engineering students: an experience report. *Comput. Sci. Educ. 28*, 1 (2018), 23–41.
[21] Carvalho, V. A., and Almeida, J. P. A. Toward a well-founded theory for multi-level conceptual modeling. *Software & Systems Modeling 17* (2018), 205–231.
[22] de Lara, J., and Guerra, E. Deep Meta-modelling with METADEPTH. In *the 48th International Conference on Objects, Models, Components, Patterns* (2010).
[23] de Lara, J., Guerra, E., and Cuadrado, J. S. Model-driven engineering with domain-specific meta-modelling languages. *SoSyM 14*, 1 (2013), 429–459.

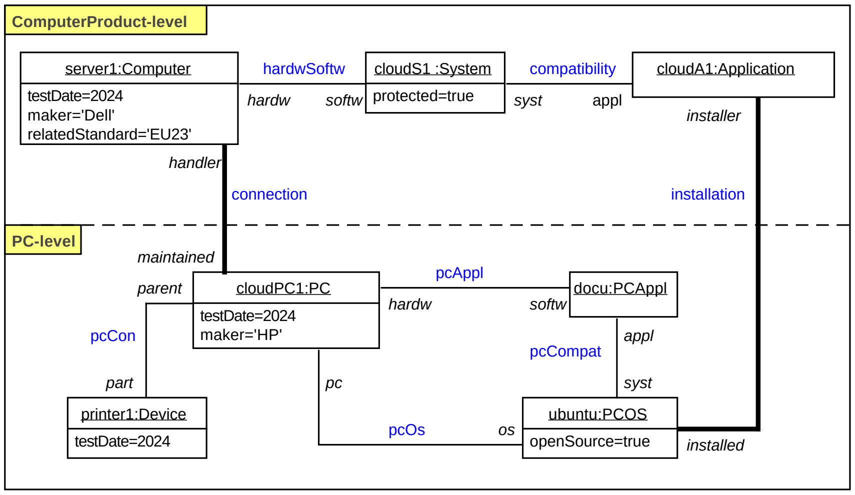


Figure 6: An illegal instance of the MedMLM-model in Figure 1

[24] de Lara, J., Guerra, E., and Cuadrado, J. S. When and how to use multilevel modelling. *ACM Trans. Softw. Eng. Methodol. 24*, 2 (2014), 12:1–12:46.

[25] FLORA-2: An Object-Oriented Knowledge Base Language. *FLORA-2 version 0.95 (Androcymbium)*, 2007.

[26] Fonseca, C. M., Almeida, J. P. A., Guizzardi, G., and Carvalho, V. A. Multi-level conceptual modeling: Theory, language and application. *Data & Knowledge Engineering 134* (2021), 101894.

[27] Frank, U. Multilevel modeling- toward a new paradigm of conceptual modeling and information systems design. *Business. & Inf. Systems Eng. 6*, 6 (2014), 319–337.

[28] Frank, U. Multi-level modeling: cornerstones of a rationale. *Software and Systems Modeling 21*, 2 (2022), 451–480.

[29] Frank, U., and Töpel, D. Contingent level classes: motivation, conceptualization, modeling guidelines, and implications for model management. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (2020), pp. 1–10.

[30] Gerbig, R., Atkinson, C., de Lara, J., and Guerra, E. A feature-based comparison of Melanee and Metadepth. In *MULTI@MoDELS* (2016).

[31] Gogolla, M., Burgueño, L., and Vallecillo, A. Model finding and model completion with USE. In *Proceedings of MODELS 2018 Workshops: ModComp, MRT, OCL, FlexMDE, EXE, COMMitMDE, MDETools, GEMOC, MORSE, MDE4IoT, MDEbug, MoDeVVa, ME, MULTI, HuFaMo, AMMoRe, PAINS co-located with ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October, 14, 2018* (2018), R. Hebig and T. Berger, Eds., vol. 2245 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 194–200.

[32] Gogolla, M., Büttner, F., and Richters, M. USE: A uml-based specification environment for validating UML and OCL. *Sci. Comput. Program. 69*, 1-3 (2007), 27–34.

[33] Jácome-Guerrero, S. P., and de Lara, J. Totem: Reconciling multi-level modelling with standard two-level modelling. *Computer Standards & Interfaces 69* (2020).

[34] Jeusfeld, M. A., and Neumayr, B. Deeptelos: Multi-level modeling with most general instances. In *ER 2016* (2016), pp. 198–211.

[35] Khitron, I., Balaban, M., and Kifer, M. FOML Site. https://goo.gl/AgxmMc, 2021.

[36] Khitron, Y. *Logic-Based Software Modeling*. PhD thesis, Computer Science Department, Ben Gurion University of the Negev, 2022.

[37] Kifer, M., Lausen, G., and Wu, J. Logical foundations of object-oriented and frame-based languages. *Journal of ACM 42* (July 1995), 741–843.

[38] Kuhlmann, M., and Gogolla, M. From UML and OCL to relational logic and back. In *Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings* (2012), R. B. France, J. Kazmeier, R. Breu, and C. Atkinson, Eds., vol. 7590 of *Lecture Notes in Computer Science*, Springer, pp. 415–431.

[39] Kühne, T. Exploring potency. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems* (2018), pp. 2–12.

[40] Lange, A., and Atkinson, C. Multi-level modeling with melanee. In *MULTI 2018 Workshop co-located with MODELS 2018* (2018), pp. 653–662.

[41] Lange, A., and Atkinson, C. On the rules for inheritance in lml. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)* (2019), pp. 113–118.

[42] Liskov, B. H., and Wing, J. M. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst. 16*, 6 (1994), 1811–1841.

[43] Macías, F., Rutle, A., Stolz, V., Rodríguez-Echeverría, R., and Wolter, U. An approach to flexible multilevel modelling. *Enterprise Modelling and Information Systems Architectures 13* (2018), 10:1–10:35.

[44] Muzaffar, I., Grossmann, G., Selway, M., and Stumptner, M. An integrated multi-level modeling approach for industrial-scale data interoperability. *Software & Systems Modeling 17*, 1 (2018), 269–294.

[45] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M. Telos: Representing knowledge about information systems. *ACM TOIS 8*, 4 (1990), 325–362.

[46] Neumayr, B., Grün, K., and Schrefl, M. Multi-level domain modeling with m-objects and m-relationships. In *APCCM 2009* (2009).

[47] Neumayr, B., Jeusfeld, M., Schrefl, M., and Schütz, C. Dual deep instantiation and its conceptbase implementation. In *Intl. Conf. on Advanced Information Systems Eng.* (2014), pp. 503–517.

[48] Neumayr, B., Schuetz, C. G., Jeusfeld, M. A., and Schrefl, M. Dual deep modeling: MLM with dual potencies and its formalization in F-Logic. *SoSyM* (2016).

[49] Selway, M., Stumptner, M., Mayer, W., Jordan, A., Grossmann, G., and Schrefl, M. A conceptual framework for large-scale ecosystem interoperability and industrial product lifecycles. *Data Knowl. Eng. 109*, C (May 2017), 85–111.

[50] Somogyi, F. A., Mezei, G., Theisz, Z., Bácsi, S., and Palatinszky, D. Playground for multi-level modeling constructs. *Software and Systems Modeling 21*, 2 (2022), 481–516.