Towards Flexible, Incremental, and Paradigm-agnostic Consistency Checking in Multi-level Modeling Environments

Andreas Demuth, Markus Riedl-Ehrenleitner, and Alexander Egyed

Institute for Software Systems Engineering,
Johannes Kepler University Linz, Austria
{firstname.lastname}@jku.at
http://www.jku.at/isse

Abstract. Multi-level modeling has become a popular paradigm as it allows for a natural and easy-to-understand representation of various real-word hierarchies. To date, several approaches have been proposed on how multi-level models should be represented and constructed – however, their continuous evolution and consistency has received considerably less attention. Consistency checking is critical to efficient and effective modeling—especially to understand the impact of model changes. Multi-level modeling adds another dimension because it allows for both model and metamodel changes over multiple levels. This paper discusses the key challenges for consistency checking in multi-level modeling environments and outlines an incremental and highly flexible approach for addressing these challenges effectively without being limited to a specific modeling paradigm. A prototype implementation of the approach has been developed; preliminary evaluation results suggest that the approach scales and provides instant consistency information during multi-level modeling.

1 Introduction

By applying model-driven engineering (MDE) approaches, practitioners raise the level of abstraction in software and systems engineering. This allows for easier communication and more efficient development processes as models become first class development artifacts that are used as blueprints for (semi-)automatic generation of the desired system. However, it has been shown that traditional two-layer approaches, in which a domain-specific language is used to model a specific instance of the domain, suffers from a lack of support for expressing the often complex hierarchies that occur in real-world domains. Even though there exist workarounds for handling such hierarchies in common two-layer modeling languages and tools (e.g., in UML), the solutions are usually not generic and often counter-intuitive. Multi-level modeling allows for modeling arbitrary deep hierarchies by allowing specific models to serve both as domain-specific instance models of a certain domain and the domain language of another instance model—a more specialized domain. Although it has been shown that multi-level modeling

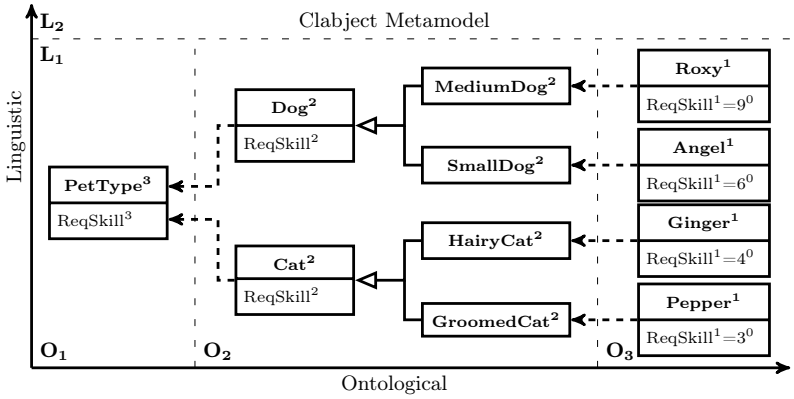

Fig. 1: Pet Store Web Shop Example (based on [4]).

makes models more intuitive to construct and read, it is still an open question in the research community how exactly model construction should be done and which methods should be used. Therefore, to date there exist various multi-level modeling approaches, paradigms, and tools (e.g., [1–3]). However, there is an important aspect that has been widely overlooked in multi-level modeling so far: the need for consistency checking in models. While it is well acknowledged that in MDE consistency checking is a crucial factor for building valid models efficiently, this has not been addressed sufficiently in multi-level modeling approaches. Although some approaches do define well-formedness constraints, these constraints typically define the semantics for a specific modeling paradigm only; there is usually no support for user-defined, domain-specific consistency rules.

In this paper, we outline the dimensions of consistency checking in multi-level modeling environments and present a generic and paradigm-agnostic approach that addresses these challenges, allowing modelers to easily write domain-specific consistency rules that check both syntax and semantics. While the approach is based on the general principles of incremental consistency checking for traditional two-level models, these concepts have been adapted and extended in order to handle multi-level models efficiently.

2 Multi-level Modeling Example

To illustrate the issues of consistency checking in multi-level modeling environments we use a standard example which is well known in the multi-level modeling community: the pet store web shop ontology [4]. As shown in Fig. 1, the example is modeled using the paradigm of *clabjects* [2] (the actual clabject metamodel is omitted for space reasons). Therefore, there are two levels in the linguistic dimension: the clabject metamodel at level $\mathtt{L}_2$ and the instance model at level $\mathtt{L}_1$ in which clabjects are used to model various ontological levels. In the ontological

dimension, there are three levels. The level O_1 defines the concept of a PetType, which has an attribute named ReqSkill which indicates the required skill level a prospective owner of a specific pet should exhibit in order to take good care of it. Different types of pets are then defined at level O_2. Specifically, the two types Dog and Cat are defined. For each of these two types, two subtypes (or specializations) are defined through inheritance: MediumDog and SmallDog for the pet type Dog as well as HairyCat and GroomedCat for the pet type Cat. Finally, for each specialized kind of dog or cat, there is a single animal available: Roxy, Angel, Ginger, and Pepper. Since these animals are instances of the specialized kinds, they are modeled at level O_3.

3 Dimensions of Consistency Checking in Multi-level Modeling

Let us now discuss the different dimensions of consistency checking that are required in the example. Specifically, there are three major dimensions: i) linguistic conformance, ii) ontological conformance, and iii) evolution.

3.1 Checking Linguistic Conformance

As shown in Fig. 1, the example spans across two linguistic levels: L_1 and L_2. All model elements, regardless of the ontological level they reside on, must conform to the metamodel defined in L_2. In the example, the metamodel to which all model elements at L_1 must conform is that of the clabject modeling paradigm. Therefore, at L_1 it must be checked whether model elements are syntactically correct and whether they obey clabject semantics. For example, a syntactic consistency rule would be that every model element (e.g., the clabject Dog) must have a name and a potency assigned. This potency must be reduced by 1 with every instantiation (e.g., the clabject Dog must have a potency of 2 because it is an instance of PetType, which has a potency of 3)—this is an example for a consistency rule checking modeling-paradigm-specific semantics. A formalization of these syntax and semantics rules that could be checked by a standard consistency checker, is depicted in Listing 1.1 (lines 1–4). The rule is written in OCL. However, note that checking semantics is necessary regardless of the used modeling paradigm; it would also be necessary if, for instance, the example was modeled with *powertypes* [1].

3.2 Checking Ontological Conformance

The second dimension we discuss is that of ontological conformance. In particular, this dimension has three major areas of interests: level-specific syntax and semantics, weak typing, and the handling of advanced modeling concepts such as inheritance.

```
1 context Clabject inv:
2 self.name<>null and
3 self.potency<>null and
4 self.potency = type.potency-1;
5
6 context PetType inv: self.ReqSkill>=0 and self.ReqSkill<=10;
7 context Dog inv: self.ReqSkill>=5;
8 context Cat inv: self.ReqSkill>=3;
9 context MediumDog inv: self.ReqSkill>=7;
```

Listing 1.1: Consistency Rules for Linguistic and Ontological Conformance.

Level-specific Syntax and Semantics. This involves checking whether a model at a given ontological level is semantically and syntactically conforming to its ontological metamodel (i.e., to its parent ontological level). Since domain-specific syntax rules do not differ significantly from linguistic syntax rules (e.g., lines 1–4 of Listing 1.1), we omit a detailed discussion here for space reasons. An example for domain-specific semantics could be the field `ReqSkill`, originally defined in `PetType` at level $\mathtt{O}_1$, which must remain within a range of `0`–`10`. A value of `0` indicates that the animal does not need any care at all and `10` indicates that the animal requires extensive care on a daily basis. A corresponding OCL consistency rule is shown in 1.1 (line 6). It must be ensured that all model elements at level $\mathtt{O}_3$ have an appropriate value set.

However, there might be additional semantic rules added at level $\mathtt{O}_2$. For example, the range of skill levels required for handling a dog should be greater than or equal to `5` because dogs must be walked at least twice a day and they also tend to adopt undesired behavioral patterns if not handled correctly. For cats, on the other hand, the required skill level should be no smaller than `3` as they require, for instance, feeding at a regular basis. The corresponding OCL consistency rules are shown in Listing 1.1 (lines 7 and 8, respectively).

Moreover, there might be more specific requirements for certain kinds of dogs and cats. For instance, medium sized dogs may have a minimum skill level of `7` because they are harder to keep under control than small dogs due to their increased strength compared to small dogs. This is expressed in the rule shown in Listing 1.1 (line 9). Again, these domain-specific semantics have to be enforced at level $\mathtt{O}_3$.

Therefore, different semantics are defined at different ontological levels. Depending on the followed modeling paradigm, it might be possible that each ontological level defines its own semantics, or each level might only refine the semantics defined at the levels above. Either way, it is required that at each ontological level conformance rules regarding syntax and semantics can be defined—which is typically not possible with existing consistency checking approaches.

Weak Typing. In multi-level modeling paradigms, type hierarchies across ontological levels are usually modeled by using concepts defined in the linguistic level $\mathtt{L}_2$. Typically, references between model elements are used to model instantiations and similar relations. For instance, the type `Clabject` may have a

reference `instanceof` that points to another `Clabject` and models instantiations. However, similar concepts exist in most—if not all—multi-level modeling paradigms. In order to write consistency rules for individual ontological levels, it is necessary to use these references to discover the ontological type of an element. Unfortunately, consistency checkers often work with linguistic types rather than with ontological types (i.e., they use runtime type information of objects). Thus, they are only capable of checking linguistic conformance (e.g., they may only check instances of `Clabject`, but not modeled instances of `Cat`).

Advanced Modeling Concepts. Similar to weak typing, there are advanced modeling concepts such as inheritance that are typically only handled at the linguistic level by consistency checkers. For example, above we discussed the semantics constraint that dogs require a minimum skill level of `5`. Indeed, this should be checked not only for direct instances of `Dog`, but also for instances of the defined subtypes `MediumDog` and `SmallDog` (i.e., `Roxy` and `Angel`). However, a standard consistency checker would not be able to handle such modeled inheritance (similar to modeled instantiation) as it would typically only consider inheritance at the linguistic level (e.g., if at the level $\mathtt{L}_2$ there was a specialization of `Clabject`, semantics defined for standard clabjects would also be checked for instances of the specialized clabjects). Moreover, at different ontological levels there might be different understandings of inheritance and thus different semantics attached, and at some levels it might be undesired to have available such modeling concepts at all. Thus, relying on a single understanding of inheritance that is defined at the top linguistic level—again, regardless of the actual paradigm used—is insufficient; it is required to support the explicit definition of concepts such as inheritance at ontological levels.

3.3 Handling Evolution

The third dimension that must be considered when checking consistency in multi-level modeling environments is evolution. While handling evolution is also necessary when checking traditional two-level models, multi-level modeling allows for evolution scenarios that are usually not present in two-level environments. Specifically, these scenarios are: i) dynamic type changes, and ii) dynamic changes of inheritance-relations. The scenarios can occur in multi-level modeling because of the weak typing and the flexible handling of concepts such as inheritance that is used for modeling ontological levels.

Let us consider a change in an inheritance-relation in our pet store example: the type `SmallDog` could be modeled as a specialization of `MediumDog` instead of `Dog` by simply changing the target of the inheritance reference. This would mean that model elements of the type `SmallDog` at $\mathtt{O}_3$ must not only conform to semantic rules defined for instances of the types `Dog` and `SmallDog`, but also to rules defined for instances of the type `MediumDog`. Moreover, note that in multi-level modeling the ontological type of a model element can be changed quite easily. The ontological type of `Angel`, for example, could be changed from

SmallDog to HairyCat by just changing the corresponding reference. This would mean that of the previously described rules only the allowed range of the required owner skill level (defined for PetType at level O_1) would be applicable to Angel.

Generally, for both change scenarios, syntax and semantics that a model element must conform to may change (i.e., a changed set of rules must be applied by a consistency checker). Such changes may be performed quite frequently due to the low cost and the typical way of how models are constructed by engineers. Therefore, it is crucial that these changes are handled efficiently—engineers typically expect modeling tools to immediately provide feedback after changes in a model have been performed.

3.4 Existing Support for Consistency Checking in Multi-level Modeling

To date, there exists a variety of consistency checking approaches (e.g., [5–11])—we now briefly summarize how they support the three dimensions of consistency checking in multi-level modeling environments. Generally, linguistic conformance can be checked sufficiently. Some approaches also handle evolution efficiently (e.g., [5]). However, checking ontological conformance and handling dynamic changes of types and inheritance at the ontological level is typically not supported by existing approaches. This is especially the case when requiring user-definable consistency rules.

4 Consistency Checking in Multi-level Modeling Environments

To address the issue of missing support for the dimension of ontological conformance checking, including the efficient handling of weak typing and dynamic changes of inheritance, we propose a novel approach to consistency checking in multi-level modeling environments that relies on a unification of linguistic and ontological levels. The approach allows for arbitrary and domain-specific consistency rules to be defined and applied at all modeled levels. It is paradigm-agnostic, thus supporting any multi-level modeling paradigm.

4.1 Linguistic and Ontological Dimension Unification

The cornerstone of our approach is the unification of ontological and linguistic levels. This allows a consistency checker to be employed for checking linguistic and ontological conformance alike. To achieve unification, the multi-level modeling paradigm, which is defined at level L_2, is transformed to a model at the newly added ontological level O_0. Thus, in our approach all levels of interest are of ontological nature, with the used multi-level modeling paradigm being the top-most ontological level. The result of this transformation for our running example is shown in Fig. 2. Again, please note that we use clabjects in the illustration but any modeling paradigm is supported at O_0. For modeling L_1 (i.e., O_0–O_3 in Fig. 2),

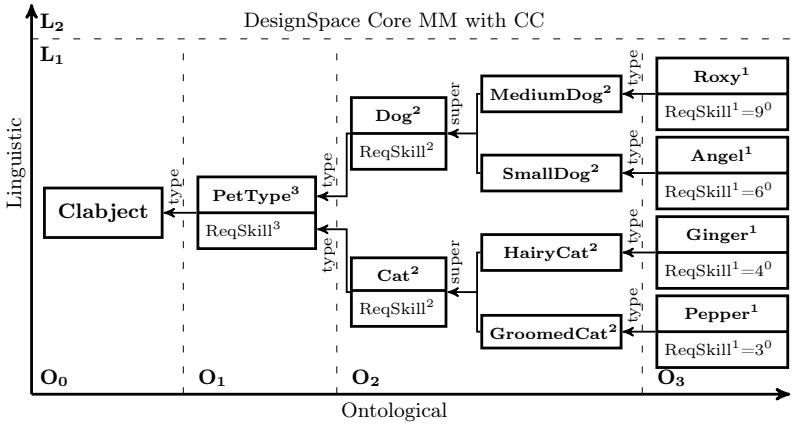

Fig. 2: Paradigm-agnostic Multi-level Modeling Scenario in the DesignSpace.

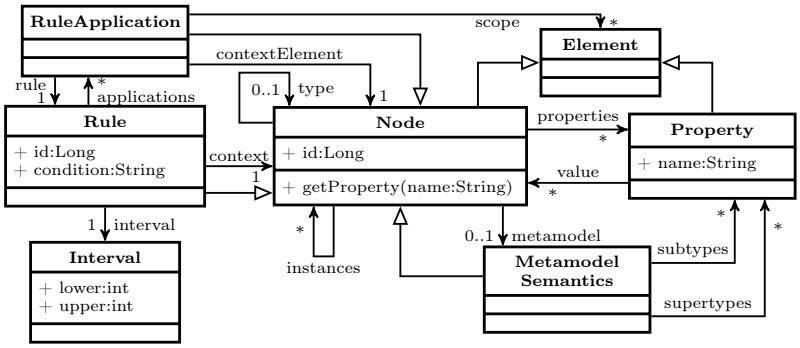

Fig. 3: DesignSpace Core Metamodel with Multi-level Consistency Checking.

which differs from $\mathtt{L_1}$ in Fig. 1, a new metamodel—called the *DesignSpace Core Metamodel (DSCM)*—is used that has been defined specifically for the purpose of flexible, multi-level modeling with consistency checking. This metamodel, which is depicted in Fig. 3, is based on a subset (hence "Core") of the metamodel for flexible modeling used in our previous work on the *DesignSpace* [12] modeling environment that was enhanced with concepts to support multi-level consistency checking.

The DesignSpace Core Metamodel is sufficiently generic to model arbitrary data structures using the simple concept of nodes (type `Node`) that can provide named properties (type `Property`). Instantiation is modeled using the reference `type`. The modeled instances of a node can be retrieved through the reference `instances`. The DSCM can be used to model any multi-level modeling paradigm at the level $\mathtt{O_0}$. In Fig. 2, the running example from Fig. 1 is modeled using DesignSpace concepts (simplified for readability reasons).

Note that consistency rules (type `Rule`), which can be used to check both syntax and semantics constraints (attribute `condition`), can be defined for any node (reference `context`), regardless of its ontological level. Our approach follows the principle of incremental consistency checking (e.g., [5]). There is a specific type (`RuleApplication`) that is used to model an individual application of a consistency rule. However, the way how rules are applied in a multi-level modeling environment differs significantly from two-level modeling approaches. Moreover, the metamodel must allow for the dynamic definition of (ontological) metamodel semantics. We will discuss these two aspects next, beginning with rule application strategies.

4.2 Rule Application Strategies

Applying consistency rules in multi-level models differs significantly from two-level models. In two-level models, rules are typically defined for a metamodel element and are then applied for all instances of that element. For example, a rule defined for the metamodel element `Clabject` in Fig. 1, such as the one defined on Listing 1.1, is applied to every single instance; i.e., every element at level $\mathtt{L_1}$. For multi-level modeling, this strategy is no longer sufficient due to the existence of multiple ontological levels and advanced rule application strategies are required.

Recall the semantics we discussed in Section 3 and the corresponding consistency rules shown in Listing 1.1 (lines 6–9), where the valid range of required owner skill levels was defined for different model elements (e.g., `PetType` at $\mathtt{O_1}$ and `Dog` at $\mathtt{O_2}$) and checked at level $\mathtt{O_3}$. Because of the used clabject modeling paradigm, checking rules that are defined at an arbitrary ontological level x at the level $x + 1$ does not make sense—the distance between the ontological level at which the rule is defined and at which it is applied can vary.

Moreover, note that there may be rules that should be checked not only at a single level, but at multiple levels—depending on the modeling paradigm. While the clabject paradigm requires attributes to have actual values assigned if, and only if, the attribute's potency is `1`, other paradigms may require that an attribute has a value assigned starting with a certain ontological level y. If the type containing the attribute is then, for example, refined at level $y + 1$, the attribute must still be set, yet it may have a different value assigned. Thus, the rule should be applied at the levels within the range of $[y; y + 1]$. Generally, it should therefore be possible to define for a rule a range of levels at which it should be checked (i.e., an interval $[a; b]$ where $a, b \in \mathbb{N}^+$).

Finally, there might be consistency rules that are defined at an ontological level z and that should be applied at all subsequent levels $z + i$ where $i \in \mathbb{N}^+$. Therefore, there should be the possibility of defining rule application ranges such as $[z + 1; \infty]$. Note that using such ranges semantically makes the ontological level at which the rule is defined (i.e., z) semantically to a linguistic level. In our example in Fig. 2, it is possible to check linguistic conformance to the modeling paradigm by defining consistency rules that express paradigm semantics at level $\mathtt{O_0}$ and using a rule application range of $[1; \infty]$.

Our approach supports all discussed rule application strategies. The strategy can be choosen for each consistency rule individually, typically depending on the used modeling paradigm. In the DSCM, rule application strategies are defined using the type `Interval`. Similar to cardinalities in UML, using `-1` instead of a positive integer allows the definition of an unbounded interval (e.g., $[1; -1]$).

4.3 Definition of Metamodel Semantics

As we have discussed above, advanced modeling concepts such as inheritance may be realized differently at different ontological levels. To support level-specific semantics, the DSCM included the type `MetamodelSemantics`. Specifically, for any node the metamodel semantics can be specified to define which properties are used to identify the node's respective super- and subtypes. For example, in Fig. 2 the property `super` models inheritance at $\mathtt{O}_2$. This allows consistency checkers to use a generic mechanism to dynamically discover metamodel semantics at any ontological level, and it enables dynamic changes of metamodel semantics at all times in flexible modeling tools. The latter is also beneficial for supporting incremental checking of constraints.

4.4 Efficient Evolution Handling

A key feature of multi-level modeling is the flexibility modelers have during modeling. Not only may modeled instances change, but also metamodels may change at all times. Therefore, it is of crucial importance that dynamic changes (e.g., type changes, changes in inheritance hierarchy) are processed efficiently so that modelers get immediate feedback. The DSCM in Fig. 3 includes the core concepts of incremental consistency checking [5] which were adapted for supporting multi-level modeling scenarios. In particular, notice the type `RuleApplication` which models a specific validation of a consistency rule on a specific instance (the `contextElement`) of the validated rule's `context`. During such a validation, a `scope` is built that contains all elements relevant by any means for the validation (i.e., any element that was accessed). This scope is used to find affected rule application whenever evolution takes place. A detailed explanation of the concept can be found in [5] and we omit a detailed discussion for space reasons. However, note that—in contrast to standarad consistency checking approaches—type-hierarchies and metamodel semantics are also `Element`s that are accessed dynamically when searching for locations to apply rules or when searching the rules to be applied to a specific model element. Thus, types and metamodel semantics can be part of rule application scopes. This means that dynamic type or inheritance hierarchies changes can also be handled efficiently (i.e., it can be determined easily which rule applications may be affected after such a change).

4.5 Prototype Implementation

A prototype implementation of the approach has been developed that is based on the DesignSpace [12] modeling framework and an adapted version of the Model/-Analyzer [5] consistency checker. A preliminary performance analysis suggests

that required adaptations for multi-level modeling (e.g., rule application mechanism) do not impose performance drawbacks compared to the two-level version of the consistency checker. However, a detailed analysis of the performance effects is part of future work.

5 Conclusions and Future Work

In this paper we have discussed the dimensions of consistency checking in multi-level modeling and outlined a paradigm-agnostic approach that handles these dimensions and provides the well-known advantages of incremental consistency with domain-specific, user-definable rules. A prototype implementation of the approach demonstrated its feasibility. A complete validation of the approach, including scalability and applicability studies as well as a detailed description of key algorithms for handling changes unique to multi-level modeling, will be done in future work.

Acknowledgments

The research was funded by the Austrian Science Fund (FWF): P25289-N15 and P25513-N15, and the Austrian Center of Competence in Mechatronics (ACCM): Strategic Research Grant C210101

References

1. J. Odell, "Power types," *JOOP*, vol. 7, no. 2, pp. 8–12, 1994.
2. C. Atkinson, "Meta-modeling for distributed object environments," in *EDOC*, pp. 90–101, 1997.
3. C. Atkinson and T. Kühne, "Processes and products in a multi-level metamodeling architecture," *International Journal of Software Engineering and Knowledge Engineering*, vol. 11, no. 6, pp. 761–783, 2001.
4. C. Atkinson, R. Gerbig, and B. Kennel, "On-the-fly emendation of multi-level models," in *ECMFA*, pp. 194–209, 2012.
5. A. Egyed, "Automatically detecting and tracking inconsistencies in software design models," *IEEE Trans. Software Eng.*, vol. 37, no. 2, pp. 188–204, 2011.
6. C. Nentwich, W. Emmerich, and A. Finkelstein, "Consistency management with repair actions," in *ICSE*, pp. 455–464, 2003.
7. C. K. F. Corrêa, "Towards automatic consistency preservation for model-driven software product lines," in *SPLC Workshops*, p. 43, 2011.
8. M. A. A. da Silva, A. Mougenot, X. Blanc, and R. Bendraou, "Towards automated inconsistency handling in design models," in *CAiSE*, pp. 348–362, 2010.
9. S. Easterbrook and B. Nuseibeh, "Using viewpoints for inconsistency management," *Software Engineering Journal*, vol. 11, no. 1, pp. 31 –43, 1996.
10. C. Xu, S.-C. Cheung, and W. K. Chan, "Incremental consistency checking for pervasive context," in *ICSE*, pp. 292–301, 2006.
11. L. M. Rose, D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "Enhanced automation for managing model and metamodel inconsistency," in *ASE*, pp. 545–549, 2009.
12. M. Riedl-Ehrenleitner, A. Demuth, and A. Egyed, "Towards model-and-code consistency checking," in *COMPSAC*, pp. 85–90, 2014.