# Taking the Pick out of the Bunch – Type-Safe Shrinking of Metamodels

Alexander Bergmayr[1], Manuel Wimmer[2], Werner Retschitzegger[3], Uwe Zdun[4]

[1]Vienna University of Technology, Austria
[2]Universidad de Málaga, Spain
[3]Johannes Kepler University Linz, Austria
[4]University of Vienna, Austria

[1]bergmayr@big.tuwien.ac.at, [2]mw@lcc.uma.es
[3]retschitzegger@cis.jku.at, [4]uwe.zdun@univie.ac.at

**Abstract:** To focus only on those parts of a metamodel that are of interest for a specific task requires techniques to generate metamodel snippets. Current techniques generate strictly structure-preserving snippets, only, although restructuring would facilitate to generate less complex snippets. Therefore, we propose metamodel shrinking to enable type-safe restructuring of snippets that are generated from base metamodels. Our approach allows to shrink a selected set of metamodel elements by automatic reductions that guarantee type-safe results by design. Based on experiments with 12 different metamodels from various application domains, we demonstrate the benefits of metamodel shrinking supported by our prototypical implementation build on top of the Eclipse Modeling Framework (EMF).

## 1 Introduction

With the adoption of Model-Driven Engineering (MDE), more and more modeling languages are defined based on metamodels. Large metamodels such as the current UML metamodel rely typically on complex structures which are challenging to grasp. For instance, manually identifying the effective classifiers and features of a certain diagram type in the metamodel requires much effort. The UML classifier `Class` transitively inherits from 13 other classifiers and provides 52 structural features which shows that even putting the focus only on one classifier can already be challenging. Allowing one to snip out a subset of a metamodel would relieve one from the full complexity imposed by the base metamodel. For instance, this would be beneficial for automating model management tasks by formulating model transformations on metamodel subsets instead of their base metamodels [SMM+12].

However, if the extraction of an effective metamodel subset is aspired, we are not only confronted with the selection of classifiers and features of the base metamodel, but also with their reduction to actually shrink the number of classifiers or generalizations. Such reductions can be useful because the design structure of the base metamodel may not be

necessarily a good choice for the extracted metamodel subset. It has to be noted that a naive reduction of classifiers may lead to inconsistencies such as $(i)$ broken inheritance hierarchies, $(ii)$ missing feature containers, and $(iii)$ dangling feature end points which require special attention in the shrinking process.
In this work, we propose an approach to automatically shrink metamodels. The result of shrinking a metamodel is what we call a metamodel snippet. A metamodel snippet is considered as a set of metamodel elements, i.e., classifiers and features, originally defined in the base metamodel. We provide refactorings to restructure initially extracted metamodel snippets. Thereby, we enable the reduction of metamodel elements that may become obsolete by a restructuring and enhance the understandability of metamodel snippets. For instance, consider the reductions of deep inheritance hierarchies that may not be necessarily required for a metamodel snippet. Applying reductions to metamodel snippets distinguishes our approach from some recent work [SMBJ09, KMG11, BCBB11] that generate strictly structure-preserving results. Reductions enable metamodel snippets with a lower number of classifiers and features, and flatter inheritance hierarchies. Our proposed reductions are type-safe in the sense that extensional equivalence[1] between extracted and reduced metamodel snippets is guaranteed by design. Our approach relies on $4$ operators: $(i)$ *Select* to define the initial set of classifiers and features, $(ii)$ *Extract* to generate a structure-preserving metamodel snippet, $(iii)$ *Reduce* to shrink the metamodel snippet, and $(iv)$ *Package* to compose all elements into a metamodel snippet that can be used as any other metamodel.
The structure of this paper is as follows. In Section 2, we introduce our metamodel shrinking approach. A prototype has been implemented based on the Eclipse Modeling Framework[2] (EMF) that is presented in Section 3. We critically discuss the results of applying our prototype for 12 metamodels in Section 4. A comparison of our approach to related work is presented in Section 5, and finally, lessons learned and conclusions are given in Section 6.

## 2 Metamodel Shrinking

Our proposed metamodel shrinking approach relies on OMG's MOF[3] abstraction level and is therefore metamodel agnostic. A metamodel snippet $MM_{snippet}$ is produced by applying our approach to a base metamodel $MM_{base}$ as shown in Fig. 1.

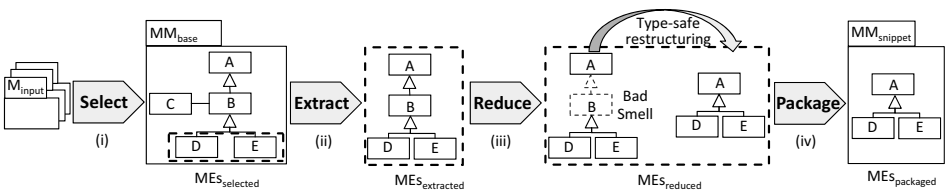


Figure 1: Overview of metamodel shrinking approach

[1] A metamodel defines a collection of models, i.e., extensions, that conform to it [VG12].
[2] `http://www.eclipse.org/modeling/emf`
[3] `http://www.omg.org/mof`

We propose a 4-step metamodel shrinking process. Each step is accompanied by a dedicated operator. The *Select* operator identifies based on a set of models $M_{input}$ all metamodel elements $MEs$, i.e., classifiers and features, required to produce these models. However, a selection of metamodel elements driven by collecting only directly instantiated classifiers may not be sufficient. Indirectly instantiated classifiers, and thus, the classifier taxonomy need to be additionally considered to end up with a valid $MM_{snippet}$. This is exactly the task of the *Extract* operator. The operator produces a set of connected metamodel elements that strictly preserves the structure of the base metamodel. Subsequently, the *Reduce* operator shrinks the result of the extraction step. To achieve a reduction of metamodel elements, we apply well-known refactorings [Opd92, HVW11] to the extracted $MM_{snippet}$. In this way, deep inheritance hierarchies without distinct subclasses are reduced. Indicators for refactorings are often referred to as 'bad smells'. For instance, in Fig. 1, we assume that class `B` 'smells bad', because it does not contain any feature for its subclasses. By removing the class and linking its subclasses directly to its superclass, the smell is eliminated. Finally, the *Package* operator serializes the reduced set of metamodel elements into a persistent metamodel. In the following subsections, we discuss these 4 steps in more detail.

### 2.1 Selection of required metamodel elements

In the selection step, all classifiers and features of interest are determined. This *explicit* set of metamodel elements shall be by all means part of the metamodel snippet. We support the selection step by allowing models as input for the *Select* operator. They serve as a basis to automatically identify the required metamodel elements to represent them. A potential model for selecting metamodel elements is sketched in Fig. 2.

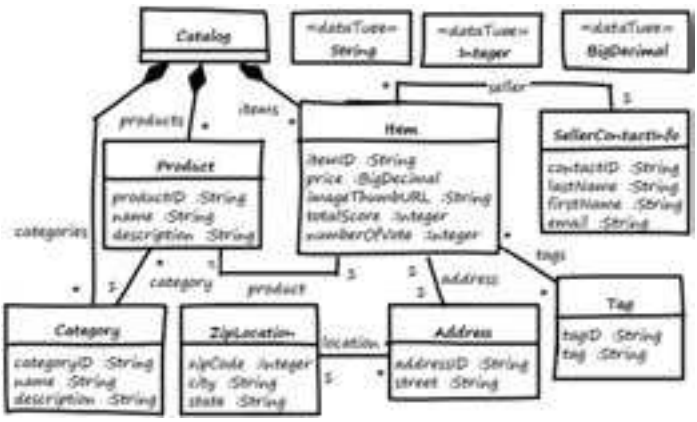


Figure 2: Class diagram of 'PetStore Navigability' application scenario

The UML class diagram shows an excerpt of the 'PetStore' scenario as introduced by Sun. The idea is to create a metamodel snippet of the UML metamodel that is effectively required

to express the 'PetStore' class diagram. We use this scenario throughout the remaining sections as a running example.

### 2.2 Extraction of selected metamodel elements

Since the explicitly selected set of metamodel elements may not be sufficient, implicit metamodel elements that glue them together need to be additionally identified. We call these elements *implicit*, as they are computed from the *explicit* set of metamodel elements produced by the *Select* operator. The *Extract* operator traverses the base metamodel and produces an enhanced set of metamodel elements by addressing $(i)$ explicitly selected metamodel elements, $(ii)$ the inheritance closure of explicitly selected classes, $(iii)$ classes that serve as container of explicitly selected inherited features, and $(iv)$ features contained by implicitly selected classes. As a result, an initial metamodel snippet is produced.
Considering the excerpt of our example in Fig. 3, `Class` was explicitly selected as the 'PetStore' class diagram contains `Class` instances. For instance, `Encapsulated-Classifier` and `StructuredClassifier` were implicitly added in addition to the explicit selection as they are in the inheritance closure of `Class`. They are considered as a means to provide a connected set of metamodel elements decoupled from the base metamodel. The decoupling is achieved by removing features of classes which reference classes not contained in the set of selected metamodel elements. In our example, 32 features were removed in the extraction step. In the reduction step, implicitly selected metamodel elements are potential candidates for becoming removed again.

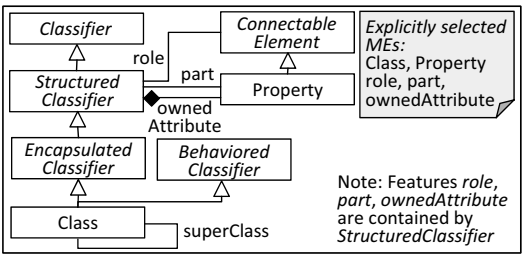


Figure 3: Extracted metamodel elements of our example

### 2.3 Reduction of extracted metamodel elements

This step aims to shrink the initial metamodel snippet. Manually identifying useful reductions is cumbersome when the number of involved metamodel elements is overwhelming and interdependencies between these reductions need to be considered. For instance, in our example, 101 metamodel elements were extracted from which 34 were reduced by applying 27 refactorings as a means to achieve a type-safe restructuring. The *Reduce* operator indicates extracted metamodel elements for reduction according to a given reduction

configuration $RC$. Such a configuration can be adapted to control the result of the *Reduce* operator. We introduce two concrete reduction configurations depicted in Fig. 4.

| | Extracted metamodel element | RC: exact | RC: extensive |
|---|---|---|---|
| Classifier | Explicit concrete `Class` | k | k |
| | Implicit concrete `Class` | k | k |
| | Explicit abstract `Class` | k | k |
| | Implicit abstract `Class` | $c^1$ | r |
| | Explicit/implicit `Datatype` | k | k |
| | Explicit/implicit `Enumeration` | k | k |
| Feature | Explicit `Feature` | k | k |
| | Implicit `Feature` | $c^2$ | $c^3$ |

*Legend for table cells:*
*RC* ... Reduction Configuration
k...keep
r...reduce
c...conditional reduce

*OCL expressions for conditions:*
fSet is assumed to be the collection of all selected features
$c^1$: **context** Class **def if** fSet->exists(f|self.ownedAttribute ->includes(f)) **then** 'k' **else** 'r' **endif**
$c^2$: **context** Feature **def if** self.lower>=1 or self.isDerived=true **then** 'k' **else** 'r' **endif**
$c^3$: **context** Feature **def if** self.lower>=1 **then** 'k' **else** 'r' **endif**

*OCL expressions for metamodel elements:*
Concrete Class: : **context** Class **inv** not self.isAbstract
Abstract Class: : **context** Class **inv** self.isAbstract

Figure 4: Exact and extensive reduction configuration (RC)

The reduction of deep inheritance hierarchies in a metamodel snippet is the rationale behind the *exact* configuration. Implicitly extracted classifiers in the shape of abstract classes, e.g., `EncapsulatedClassifier` or `StructuredClassifier` in our example, are indicated for reduction. Considering the UML metamodel, `Class` originally inherits from `EncapsulatedClassifier` which inherits from `StructuredClassifier` that in turn inherits from `Classifier`. They are all well justified in the context of the base metamodel, but may not be as important for metamodel snippets. In our example, the context was narrowed to UML's data modeling capabilities. As a result, `Encapsulated-Classifier` is indicated for reduction when applying the exact reduction configuration as shown in Fig. 5.

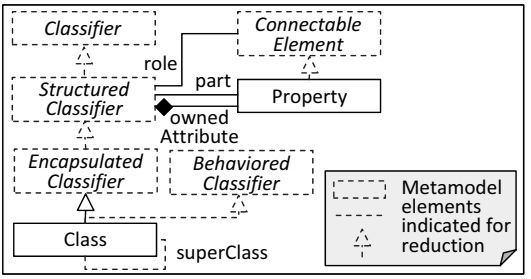


Figure 5: Metamodel elements indicated for reduction with extensive RC of our example

Similar to reducing implicitly selected classifiers, features with these characteristics are candidates for reduction except they are defined as being `required` or `derived`. While the rationale for the former exception is obvious, derived features are kept to avoid loss of information due to their calculated instead of user-defined value. The `superClass` feature of `Class` is an example in this respect.
In contrast to the exact reduction configuration, the *extensive* reduction configuration in-

dicates `derived` features for reduction. Since we did not apply UML's generalization concept for classes in our example, the `superClass` feature was reduced by the extensive reduction configuration. Rather than keeping implicitly selected abstract classes that serve as feature containers, in the extensive reduction configuration the intension is to reduce them without exceptions. Both `EncapsulatedClassifier` and `StructuredClassifier` are, thus, indicated for reduction in our example.
However, indicating metamodel elements for reduction is only half the way to obtain a useful metamodel snippet since naively reducing classes may lead to inconsistencies. We encountered three possible inconsistencies in our approach: $(i)$ broken inheritance hierarchies, $(ii)$ missing feature containers, and $(iii)$ dangling feature end points. In our example, the generalization relationship of `Class` needs to be relocated, and the feature `role` requires a new container and a new type when the indicated classes are actually reduced. To overcome these unintended effects, we conduct a type-safe restructuring by relying on well-known object-oriented refactorings [Opd92] adapted to the area of (meta)modeling [HVW11]. In Fig. 6, we introduce refactorings for the restructuring of metamodel snippets.

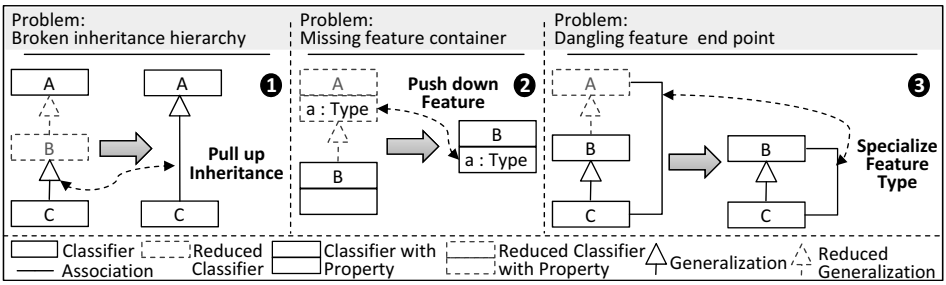


Figure 6: Refactoring techniques for type-safe metamodel restructuring

They achieve $(i)$ relocating generalization relationships by pulling up the relationship ends to super-superclasses, $(ii)$ moving features if their base containers were reduced by pushing down features from superclasses to subclasses and $(iii)$ reconnecting dangling feature end points by specializing feature types from superclasses to subclasses. These refactorings are, by design, type-safe since they operate on the inheritance hierarchies imposed by the base metamodel and respect type substitutability [GCD+12]. Refactorings are considered as events triggered by the need to usefully conduct indicated reductions on the metamodel snippet.
*Pull up inheritance.* This refactoring enables relocating generalization relationships. A relationship end that would be a dangling reference as a result of reducing classes which lie in between of other classes in the inheritance hierarchy has to be relocated. Such a gap in the inheritance hierarchy is closed by pulling up the relationship end to the least specific superclasses of the reduced class.
In our example, the generalization relationship of `Class` needs to be relocated as `Class` indirectly specializes `Classifier` and both classes are kept after the reduction. As a result, `Class` inherits from `Classifier` serving as the replacement for the more specific classes `EncapsulatedClassifier` and `StructuredClassifier` as shown in Fig. 7. The indicated reduction for `Classifier` is relaxed because several subclasses

such as `Association` or `Datatype` inherit features contained by `Classifier`. A reduction of `Classifier` would lead to duplicated features in the corresponding subclasses. We decided to prevent such an effect as from an object-oriented design perspective this is not desirable.

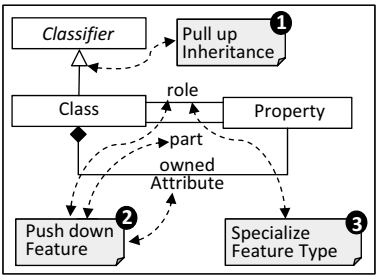


Figure 7: Refactored metamodel elements of our example

*Push down feature.* This refactoring supports moving features from one to another container by going down the inheritance hierarchy. Features for which a new container is required are moved down to the most generic subclass. This could lead to reverting back to a previously reduced container to avoid duplicated features (cf., `Classifier`). Reduced containers become in such a situation reintroduced.
In our example, the features `part`, `ownedAttribute` and `role` are moved to a container compatible with `StructuredClassifier` since this class was reduced.
*Specialize feature type.* This refactoring addresses reconnecting dangling references of associations or compositions between classes. Similar to the push down feature refactoring, the most generic subclass is selected for the type specialization.
In our example, the feature `role` needs to be reconnected to a type compatible with `ConnectableElement`. As a result, the type of feature `role` is changed from `ConnectableElement` to `Property`. Searching for the most generic subclass may lead to a similar situation like for the push down feature refactoring, i.e., the reintroduction of previously reduced classes.

### 2.4 Packaging of metamodel snippet

To enable dedicated modeling tools to work with metamodel snippets, the metamodel snippets need to be materialized. The *Package* operator takes the result of the *Reduce* operator and reconciles the shrinked set of metamodel elements into a serialized $MM_{snippet}$.
In Fig. 8, the complete result for our example is shown, i.e., the part of the UML metamodel required to express the 'PetStore' model. We applied the extensive reduction configuration which resulted in 22 classifiers and 45 features. Overall, 8 different classes were actually instantiated as indicated by the dashed framed classifiers in Fig. 8. Using only this set of classes would require to inject the same features multiple times in different classes which would lead to a metamodel snippet with poor design quality. Thus, by applying the

proposed refactorings, we can find an effective trade-off between a carefully selected set of classes and design quality.

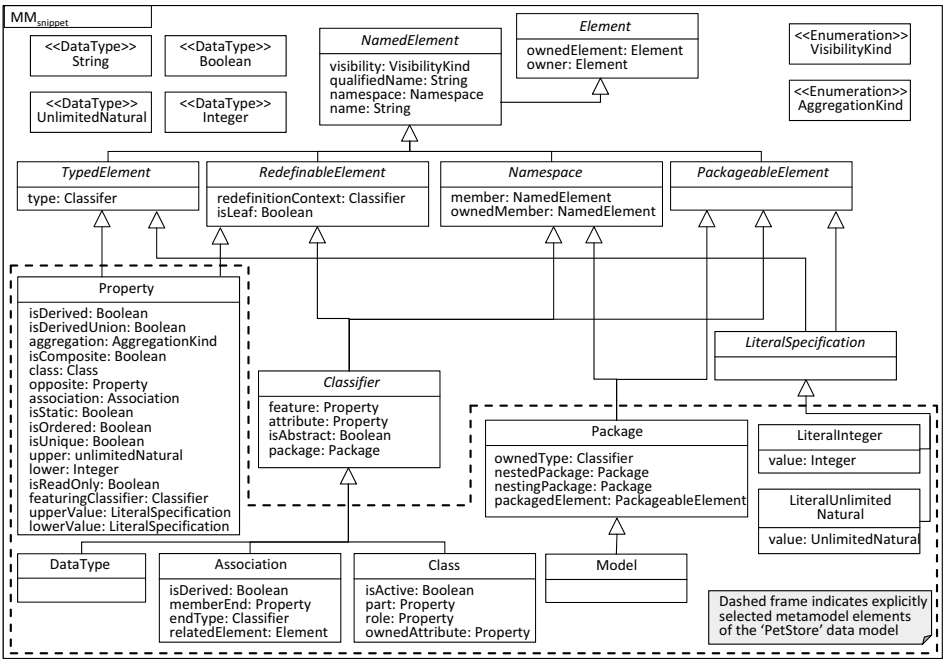


Figure 8: Metamodel snippet of our example

## 3 Prototypical Implementation: EMF Shrink

To show the feasibility of the metamodel shrinking approach, we implemented a prototype based on EMF. To operationalize our proposed operators, we implemented them based on a pipeline architecture. While the *Select* and *Extract* operator have been straightforwardly implemented on the basis of EMF, the realization of the *Reduce* operator required more care because potentially occurring side effects as a result of applied metamodel refactorings needed to be handled. An example in this respect is the reintroduction of previously reduced classes because they may have effects on the inheritance hierarchies. For that reason, we heavily exploited EMF's change notification mechanism to trigger precalculated relaxations on refactorings that become obsolete as metamodel shrinking progresses. The *Package* operator generates independently of the position in the pipeline valid metamodel snippets conforming to Ecore, i.e., EMF's meta-metamodel. This was helpful for validating and interpreting the results of our operators. We used an automatic validation by executing well-formedness constraints and manual validation by inspecting the generated snippets in the graphical modeling editor for Ecore models. Implementation code for metamodel

snippets can be generated by applying EMF's generation facility.
Customizations in a metamodel's implementation code that also relate to a metamodel snippet requires special consideration. In our running example, the value of the feature `ownedElement` in `Element` is a derived value. For that reason, we additionally realized, based on EMF's adapter concept, generic adapter factories that allow the integration of customized implementation code into generated implementation code of a metamodel snippet as far as model manipulation operations are concerned. Whenever model elements are created with a metamodel snippet, in the background corresponding model elements as instances of the base metamodel are created. As a result, model elements adapt each other in the sense of a delegation mechanism and are kept synchronized via change notifications. Further details regarding our implemented prototype can be found online[4].

## 4 Evaluation

To evaluate the applicability of our proposed approach, we performed experiments by shrinking 12 metamodels based on given models as summarized in Fig. 9.

| | Results in absolute numbers | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Packaged classifiers | | Packaged features | | Applied refactorings | |
| **Metamodel#** | Total classifiers | Total features | Extracted classifiers | Extracted features | RC1+ | RC2+ | RC1+ | RC2+ | RC1+ | RC2+ |
| ACME[1] | 19 | 26 | 7 | 4 | 5 | 5 | 4 | 4 | 2 | 2 |
| Agate[1] | 72 | 204 | 16 | 35 | 12 | 11 | 29 | 29 | 6 | 8 |
| BMM[2] | 47 | 66 | 13 | 22 | 11 | 11 | 16 | 16 | 4 | 4 |
| BPMN2[3] | 147 | 458 | 35 | 73 | 34 | 31 | 64 | 40 | 2 | 7 |
| HTML[4] | 62 | 112 | 8 | 16 | 7 | 7 | 14 | 14 | 2 | 2 |
| iStar[5] | 44 | 101 | 19 | 41 | 18 | 18 | 30 | 30 | 0 | 0 |
| PNML[6] | 42 | 80 | 14 | 16 | 10 | 10 | 16 | 16 | 2 | 2 |
| Requirement[4] | 50 | 53 | 22 | 23 | 22 | 20 | 20 | 20 | 0 | 8 |
| SBVR[4] | 332 | 361 | 20 | 18 | 20 | 20 | 11 | 11 | 0 | 0 |
| SQLDDL[4] | 20 | 27 | 13 | 21 | 13 | 13 | 20 | 20 | 0 | 0 |
| SysML[7] | 307 | 621 | 38 | 84 | 33 | 25 | 64 | 38 | 1 | 15 |
| UML[4] | 264 | 586 | 36 | 65 | 29 | 22 | 52 | 45 | 6 | 27 |
| | | | Extract | | Package | | | | Reduce | |

#)From AtlanMod Metamodel Zoo

+)RC1 ... Exact reduction configuration
+)RC2 ... Extensive reduction configuration

*Source of used models:*
[1]Order Processing System (IBM developerworks)
[2]JK Enterprises (IBM developerworks)
[3]Hardware Retail Process (OMG BPMN Specification)
[4]PetStore use case (PetStoreNavigability)
[5]Toronto Civil Workers Strike (OpenOME Toronto)
[6]Vending Machine (Workflow Petri Net Designer WoPeD)
[7]Distiller (OMG SysML)

Figure 9: Quantitative experiment results in absolute numbers

The rationale behind our selection of *Metamodels* is mainly based on three criteria: $(i)$ coverage of a wide range of applications domains (from business motivation and business process management over requirements engineering to software and systems engineering),

[4]`http://code.google.com/a/eclipselabs.org/p/emf-shrink`

$(ii)$ consideration of small-sized to large-sized metamodels (from less than $50$ to over $900$ metamodel elements), $(iii)$ involvement of metamodels with flat as well as deep inheritance hierarchies (from $2$ up to $10$ abstraction levels). *Total classifiers* and *Total features* refer to the size of a metamodel whereas *Extracted classifiers* and *Extracted features* represent the result of the extraction step in the respective experiments. Results from the *Package* operator w.r.t. the *Reduce* operator are presented for the two introduced reduction configurations. The less classifiers and features were packaged w.r.t. their corresponding number of extracted classifiers and features, the more metamodel elements were reduced. Finally, absolute numbers of applied refactorings are provided as a result of the *Reduce* operator. Typically, the more reductions of metamodel elements were achieved, the higher is the number of applied refactorings.

Based on the quantitative results, we critically discuss our approach from a qualitative perspective by investigating benefits and limitations of our approach. We consider $5$ architectural metrics (cf., [BD02, MSJ04]) of packaged compared to extracted metamodel elements: $(i)$ number of reduced classifiers, $(ii)$ number of reduced features, $(iii)$ mean features per classifier, $(iv)$ mean inheritance hierarchy depth and $(v)$ understandability.

As expected, benefits of metamodel shrinking take effect when large metamodels with deep inheritance hierarchies (cf., UML or SysML experiments) are considered. Considering Fig. 10, we could achieve to reduce in average $\approx 13\%$ of extracted classifiers with the exact reduction configuration, while an average value of $\approx 18\%$ could be achieved with the extensive one.

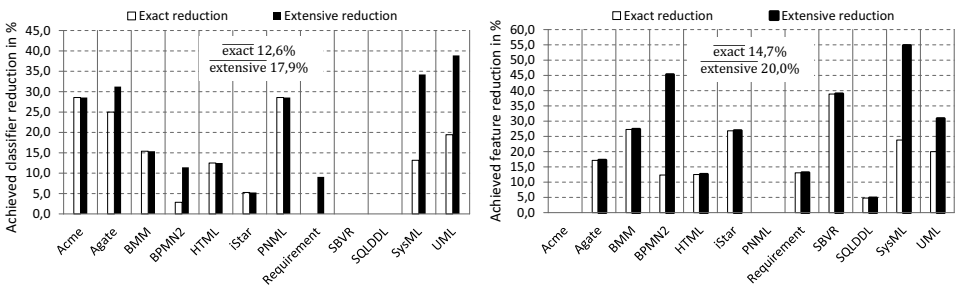


Figure 10: Achieved reductions of metamodel elements

Most classifier reductions could be achieved in the UML ($\approx 39\%$) and SysML ($\approx 34\%$) experiments as these metamodels cover many abstract classifiers for reasons of genericity or extensibility which is not necessarily required for metamodel snippets. Results of feature reductions w.r.t. the number of extracted features are in average in the range of $\approx 15\%$ to $\approx 20\%$. Reductions of features are generally easier to achieve as they lead typically not to inconsistencies in a metamodel snippet. However, in case of classifier reductions inconsistencies in a metamodel snippet may lead to reintroducing a previsouly reduced feature container to avoid duplicated features. Consequently, the rates of classifier reductions are lower than the rates of feature reductions particularly when the number of extracted features is much higher than the number of extracted classifiers.

Considering Fig. 11, the extensive reduction of classifiers may lead to an increase of mean features per classifier since features are pushed down from generic to more specific

classifiers (cf., Agate or UML experiments).

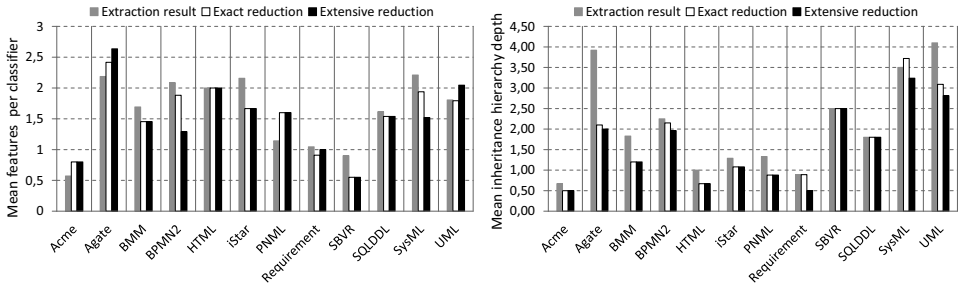


Figure 11: Mean features per classifier and mean inheritance hierarchy depth

Extensive reductions have generally positive effects on the mean inheritance hierarchy depth of metamodel snippets. The less classifiers are contained by metamodel snippets, the flatter inheritance hierarchies can be achieved while on the downside the less opportunities are available for features to be placed in appropriate classifiers. Generally, reductions of metamodel elements and potential refactorings lead inherently to structural differences between metamodel snippets and their base metamodels. Still, our metamodel shrinking approach generates metamodel snippets that are restructured in a type-safe way. Metamodel snippets enable expressing the models that were used to produce them in the same way as their base metamodels.
Finally, we applied the understandability metric of [BD02] to the metamodel snippets in our experiments as shown in Fig. 12.

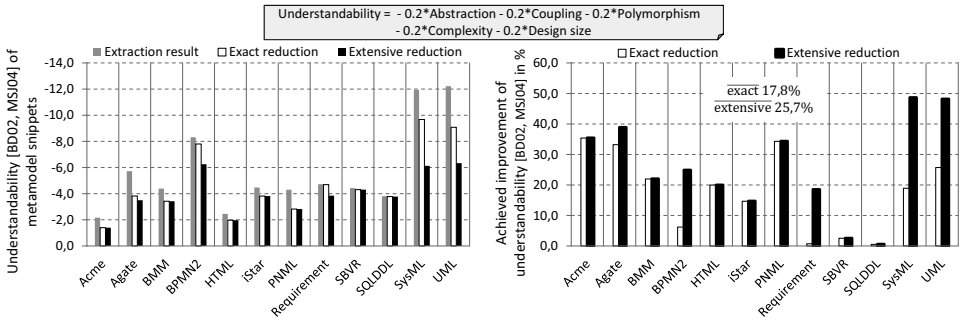


Figure 12: Understandability of metamodel snippets

We used a slightly adapted version of the originally proposed formula to calculate the understandability measures in two respects. First, we omitted the encapsulation property since private features are rarely used for metamodels, and second, we also omitted the cohesion property since our focus is on structural rather than behavioral features. As a result, our formula consists of 5 properties with equal weights, i.e., 0.2, that add up to 1 as suggested by [BD02]: $(i)$ *Abstraction*, i.e., average number of ancestors for classifiers, $(ii)$ *Coupling*, i.e., average number of features owned by a classifier that reference to other

distinct classifiers, $(iii)$ *Polymorphism*, i.e., number of abstract classifiers, $(iv)$ *Complexity*, i.e., average number of features in a classifier and $(v)$ *Design size*, i.e., number of classifiers. The calculated value of the understandability metric is negative which means the lower the value the more difficult is it to understand a metamodel.
We could improve the understandability of extracted compared to extensively reduced metamodel snippets in average by $\approx 26\%$. Considering the UML experiment, the understandability value of the UML metamodel is $\approx -61$ whereas with the *Extract* operator we could improve this value to $\approx -12$ when the focus is on UML's data modeling capabilities. With the restructuring of extracted metamodel elements, we could further improve the understandability by $\approx 48\%$ in the metamodel snippet.

## 5 Related Work

With respect to our goal of generating metamodel snippets, we identified three lines of related research work: $(i)$ model slicing, $(ii)$ model refactoring and $(iii)$ aspect mining.
*Model Slicing.* Inspired from the notion of program slicing [Wei81], a static slicing mechanism is proposed by [KMS05] for UML class diagrams and by [BLC08] for modularizing the UML metamodel. Both approaches present how (meta)model elements are selected by relying on user-defined criteria (e.g., classifiers or relationships to be included) that express the initial set of elements from which a slice is computed. This computation is in our work supported by the *Extract* operator. Slicing mechanisms specific to UML class diagrams and state machine diagrams are introduced by [LKR10, LKR11]. In this research endeavor, class invariants and pre- and post-conditions of operations are exploited for computing class diagram slices while data and control flow analysis techniques are applied to reduce state machines to the elements relevant to reach a certain state. Since in our approach metamodels are solely considered from a structural viewpoint, techniques related to behavioral viewpoints (e.g., operational semantics) are beyond the scope of our approach. Slicing metamodels is addressed by approaches presented in [SMBJ09] and more recently in [KMG11]. They apply a projection-based approach to obtain a strictly structure-preserving subset as opposed to our approach that enables restructuring of metamodels. A declarative language as a means to implement slicing mechanisms for reducing (meta)models is introduced by [BCBB11] which could be an alternative technology to realize our *Reduce* operator.
*Model Refactoring.* Existing research work in the area of model refactoring is presented by [FGSK03] addressing pattern-based refactorings on UML-based models, [Wac07] proposing model refactorings for co-adapting models with evolved metamodels, or [MMBJ09] applying generic model refactorings on different kind of models. Our approach focuses on the metamodel level. We adopted commonly known refactorings originating from the area of object-orientation [Opd92, HVW11] to achieve type-safe metamodel reductions.
*Aspect Mining.* Since aspect-orientation has arrived at the modeling level, several research endeavors started addressing this topic as surveyed in [WSK$^+$11]. Identifying aspects in existing models is investigated by [ZGLT08], presenting approaches for mining crosscutting concerns in a given set of models and describing them with an aspect language. Metamodel snippets can be compared with the notion of symmetric concerns [HOT02, WSK$^+$11] since

they subsume model elements related to certain modeling concerns.

## 6 Lessons Learned

We now summarize lessons learned from realizing and applying our proposed approach.
*Type-safe restructuring as enabler for less complex metamodel snippets.* Strictly structure-preserving approaches are not necessarily the first choice for generating metamodel snippets since restructuring can reduce the number of metamodel elements. Approaches that facilitate to compose (cf., [WS08]), combine (cf., [Val10]) or extend (cf., [LWWC12]) existing metamodels may benefit from reduced metamodel snippets if they intend to operate on a subset of a large metamodel.
*Usage of existing metamodel implementation code for metamodel snippets.* Decoupling metamodel snippets from their base metamodel requires care if existing metamodel implementation code is available. Clearly, this depends on the metamodeling workbench. We realized delegation mechanisms that loosely couple metamodel snippets with their base metamodel at the implementation level to cope with this challenge.
*Metamodel snippets as reusable assets for new metamodels.* Since reuse allows exploiting domain knowledge already expressed in existing metamodels [KKP+09], metamodel snippets can support reuse when only some parts of an existing metamodel are required. For instance, the language workbench challenge 2012[5] refers to such a reuse scenario. Clearly, metamodel snippets are a first step in this direction and may act as stimulator to enhance reuse in metamodeling.

## References


[BCBB11] Arnaud Blouin, Benot Combemale, Benoit Baudry, and Olivier Beaudoux. Modeling Model Slicers. In *MODELS'11*, pages 62–76. Springer, 2011.

[BD02] Jagdish Bansiya and Carl G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Trans. Softw. Eng.*, 28(1):4–17, 2002.

[BLC08] Jung Ho Bae, KwangMin Lee, and Heung Seok Chae. Modularization of the UML Metamodel Using Model Slicing. In *ITNG'08*, pages 1253–1254. IEEE, 2008.

[FGSK03] R. France, S. Ghosh, E. Song, and D.K. Kim. A metamodeling approach to pattern-based model refactoring. *IEEE Softw.*, 20(5):52–58, 2003.

[GCD+12] Clément Guy, Benoît Combemale, Steven Derrien, Jim R. H. Steel, and Jean-Marc Jézéquel. On model subtyping. In *ECMFA'12*, pages 400–415. Springer, 2012.

[HOT02] W. Harrison, H. Ossher, and P. Tarr. Asymmetrically vs. Symmetrically Organized Paradigms for Software Composition. Research Report RC22685, IBM, 2002.

[HVW11] Markus Herrmannsdoerfer, Sander D. Vermolen, and Guido Wachsmuth. An extensive catalog of operators for the coupled evolution of metamodels and models. In *SLE'11*, pages 163–182. Springer, 2011.


[5] `http://www.languageworkbenches.net`

[KKP+09] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel. Design Guidelines for Domain Specific Languages. In *OOPSLA'09 Workshop on Domain-Specific Modeling (DSM'09)*, 2009.

[KMG11] Pierre Kelsen, Qin Ma, and Christian Glodt. Models within Models: Taming Model Complexity Using the Sub-model Lattice. In *FASE'11*, pages 171–185. Springer, 2011.

[KMS05] Huzefa Kagdi, Jonathan I. Maletic, and Andrew Sutton. Context-Free Slicing of UML Class Models. In *ICSM'05*, pages 635–638. IEEE Computer Society, 2005.

[LKR10] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. Slicing of UML Models Using Model Transformations. In *MODELS'10*, pages 228–242. Springer, 2010.

[LKR11] Kevin Lano and Shekoufeh Kolahdouz-Rahimi. Slicing Techniques for UML Models. *JOT*, 10:1–49, 2011.

[LWWC12] Philip Langer, Konrad Wieland, Manuel Wimmer, and Jordi Cabot. EMF Profiles: A Lightweight Extension Approach for EMF Models. *JOT*, 11(1):1–29, 2012.

[MMBJ09] Naouel Moha, Vincent Mahé, Olivier Barais, and Jean-Marc Jézéquel. Generic Model Refactorings. In *MODELS'09*, pages 628–643. Springer, 2009.

[MSJ04] Haohai Ma, Weizhong Shao, Lu Zhang 0023, and Yanbing Jiang. Applying OO Metrics to Assess UML Meta-models. In *UML'04*, pages 12–26. Springer, 2004.

[Opd92] William F. Opdyke. *Refactoring object-oriented frameworks*. PhD thesis, 1992.

[SMBJ09] Sagar Sen, Naouel Moha, Benoit Baudry, and Jean-Marc Jézéquel. Meta-model Pruning. In *MODELS'09*, pages 32–46. Springer, 2009.

[SMM+12] Sagar Sen, Naouel Moha, Vincent Mahé, Olivier Barais, Benoit Baudry, and Jean-Marc Jézéquel. Reusable model transformations. *SoSym*, 11(1):111–125, 2012.

[Val10] Antonio Vallecillo. On the Combination of Domain Specific Modeling Languages. In *ECMFA'10*, pages 305–320. Springer, 2010.

[VG12] Antonio Vallecillo and Martin Gogolla. Typing Model Transformations Using Tracts. In *ICMT'12*, pages 56–71. Springer, 2012.

[Wac07] Guido Wachsmuth. Metamodel Adaptation and Model Co-adaptation. In *ECOOP'07*, pages 600–624. Springer, 2007.

[Wei81] Mark Weiser. Program slicing. In *ICSE'81*, pages 439–449. IEEE Press, 1981.

[WS08] Ingo Weisemöller and Andy Schürr. Formal Definition of MOF 2.0 Metamodel Components and Composition. In *MODELS'08*, pages 386–400. Springer, 2008.

[WSK+11] Manuel Wimmer, Andrea Schauerhuber, Gerti Kappel, Werner Retschitzegger, Wieland Schwinger, and Elisabeth Kapsammer. A survey on UML-based aspect-oriented design modeling. *ACM Comput. Surv.*, 43(4):1–33, 2011.

[ZGLT08] Jing Zhang, Jeff Gray, Yuehua Lin, and Robert Tairas. Aspect mining from a modelling perspective. *Int. J. Comput. Appl. Technol.*, 31:74–82, 2008.