

## **Towards Interoperable Metamodeling Platforms: The Case of Bridging ADOxx and EMF**

Dominik Bork, Konstantinos Anagnostou, Manuel Wimmer

Appeared in:

*Advanced Information Systems Engineering - 34th International  
Conference, CAiSE 2022, Leuven, Belgium, June 6-10, 2022,  
Proceedings, pp. 479-497)*


© 2022 by Springer.

Final version available via:

[https://doi.org/10.1007/978-3-031-07472-1\\_28](https://doi.org/10.1007/978-3-031-07472-1_28)

[www.model-engineering.info](http://www.model-engineering.info)

# Towards Interoperable Metamodeling Platforms: The Case of Bridging ADOxx and EMF

Dominik Bork <sup>1</sup>[0000–0001–8259–2297], Konstantinos Anagnostou<sup>1</sup>, and  
Manuel Wimmer<sup>2</sup>[0000–0002–1124–7098]

<sup>1</sup> TU Wien, Business Informatics Group, Vienna, Austria  
`{firstname.lastname}@tuwien.ac.at`

<sup>2</sup> Johannes Kepler University Linz, CDL-MINT, Linz, Austria  
`manuel.wimmer@jku.at`

**Abstract.** Metamodeling platforms are an important cornerstone for building domain-specific modeling languages in an efficient and effective way. Two prominent players in the field are ADOxx and the Eclipse Modeling Framework (EMF) which both provide rich ecosystems on modeling support and related technologies. However, until now, these two worlds live in isolation while there would be several benefits of having a bridge to exchange metamodels and models for different purposes (e.g., reuse of features and plugins that are only available on one platform, access to additional modeler and developer communities). Therefore, in this paper, we propose first steps toward establishing interoperability between ADOxx and EMF. For this, we thoroughly analyze the metamodeling concepts employed by both platforms before proposing a bridge that enables bidirectional exchange of metamodels. We evaluate the bidirectional bridge with several openly available metamodels created with ADOxx and EMF, respectively. Moreover, we quantitatively and qualitatively analyze the bridge by an evaluation that incorporates the instantiation and use of the metamodels on both platforms. We show that the metamodels can be exchanged without information loss and similar modeling experiences with respect to the resulting models can be achieved.

**Keywords:** Metamodeling · ADOxx · EMF · Language engineering · Tool interoperability.

## 1 Introduction

Modeling languages are ubiquitous in information systems, e.g., consider the different process modeling languages, data modeling languages, or multi-viewpoint enterprise modeling languages to mention just a few prominent examples. Metamodeling platforms are an important cornerstone for building modeling languages in an efficient and effective way [7, 8]. Such metamodeling platforms provide dedicated support to specify the modeling concepts and their relationships, i.e., the abstract syntax of the modeling language, as well as dedicated support to develop the visualization of the models in terms of textual, graphical, or even hybrid languages, i.e., the concrete syntax of the modeling language. In addition, several other technologies, such as transformation engines, simulation frameworks, or code generators are provided based on the common

abstraction referred to as the meta-metamodel. Although there are standards for meta-metamodels [8], current metamodeling platforms still use different meta-metamodels based on their particular development history, user base, or targeted use cases for the hosted modeling languages.

Two prominent players in the field are ADOxx [1] and the Eclipse Modeling Framework (EMF) [26] which both provide rich ecosystems on modeling support and related technologies. However, until now, these two worlds live in isolation while there would be several benefits of having a bridge to exchange metamodels and models for different purposes. The main reason for this is that they employ different meta-metamodels. However, thoroughly investigating if these meta-metamodels share similarities is of interest as it would allow to exchange metamodels – and eventually models – between these two worlds by dedicated model transformations [27] and benefit from reuse, particular support offered by a particular platform, and reaching additional users which are operating in the other platform. As such the integration proposed in this paper not only bridges the technological spaces but also the associated modeler and developer communities, it enables the creation of powerful tool-chains that span ADOxx and EMF, and facilitates the mutual strengths while mitigating potential shortcomings.

In this paper, we propose first steps toward establishing interoperability between ADOxx and EMF concerning the abstract syntax of the modeling languages, so to speak the foundation of the languages. For this, we thoroughly analyze the metamodeling concepts employed by both platforms before proposing a bridge that enables bidirectional exchanges of metamodels. We evaluate the bidirectional bridge with several openly available metamodels created with ADOxx and EMF, respectively. Moreover, we quantitatively and qualitatively analyze the bridge by an evaluation that incorporates the instantiation and use of the metamodels on both platforms. Our results show that the metamodels can be exchanged without information loss and also pragmatic issues such as providing the same modeling experiences with respect to the instantiated models can be achieved.

The rest of the paper is structured as follows. The foundations of metamodeling, ADOxx, and EMF are introduced in Section 2. Related work on bridging metamodeling platforms is presented in Section 3, before Sections 4 and 5 describe our approach of bridging ADOxx and EMF in detail. Section 6 evaluates our approach based on different characteristics. Finally, we conclude this paper in Section 7 and elaborate on future research directions.

## 2 Foundations

In this section, we provide the foundations for this work: *(i)* the general metamodeling architecture, and *(ii)* the two metamodeling platforms ADOxx and EMF.

### 2.1 Metamodeling

Metamodeling platforms support the development of modeling tools by providing an abstract meta-metamodel that is adequate even for non-programmers to specify (domain-specific) metamodels – and corresponding modeling tools – by modeling modeling languages, thus metamodeling [16].

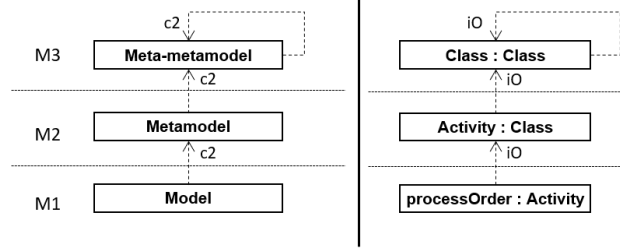


Fig. 1: Metamodeling stack: Macro view (left hand side), Micro view (right hand side)

Fig. 1 shows the metamodeling stack on a macro and micro view. Most metamodeling platforms are based on three modeling layers. Let us start with the macro view. M3 – the top layer – is providing the metamodeling language of a platform to define metamodels on M2, i.e., the modeling languages. The models build with these languages are situated on the next lower level M1. The relationships between the different artefacts on the different layers are very important. Between adjacent layers, the upper layer provides the building plan for the lower layer. Thus, we speak about *conformsTo* (c2) relationships, i.e., an artefact on layer  $n$  is valid with respect to the artefact on layer  $n + 1$ . Finally, please note that M3 is reflexive in most platforms, i.e., it is defined by itself.

The right hand side of Fig. 1 shows the micro view, i.e., looking inside the artefacts shown in the left hand side of Fig. 1. It shows a basic modeling concept called *Activity* on M2. This concept is modeled as a *class* – a concept provided on M3. Then, the activity concept is instantiated on M1 for defining the activity *processOrder*. Please note that for the micro view, we have *instance-of* (iO) relationships between the elements of different levels.

Furthermore, such platforms provide pre-configured, method agnostic functionality like model management, user management, and user interaction which are attached to abstract and generic meta-metamodel classes thereby considerably contributing to the efficient realization of modeling tools with metamodeling platforms. The language engineer only needs to adapt the platform’s meta-metamodel to her domain.

## 2.2 ADOxx

The ADOxx [1] metamodeling platform matured from industry and is nowadays widely used in academia for the development of modeling tools [7]. ADOxx is focusing on realizing graphical conceptual modeling tools with built-in generic features like graph-based model simulation and support to define model queries that can be easily customized to domain-specific modeling languages (cf. [6]). Moreover, a powerful language to realize static and dynamic graphical concrete syntaxes that go beyond tree structures is provided. ADOxx is open use and provides several extension mechanisms to plug-in or interact with third-party services and tools. In order to realize a new modeling tool with ADOxx, language engineers only need to [4]: (i) configure the specific metamodel by referring its concepts to the meta-metamodel concepts of ADOxx, optionally constrained using the platform-specific scripting language AdoScript; (ii) provide a visualization for the concepts; (iii) combine the concepts into logical chunks,

i.e., ADOxx modeltypes; and (iv) realize additional model processing functionality such as model transformations or simulations.

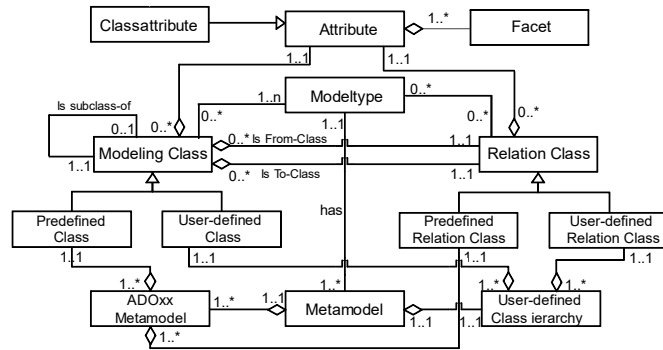


Fig. 2: Excerpt of the ADOxx dynamic meta-metamodel [5]

ADOxx metamodels are composed of *modeltypes* which themselves compose pre-defined and user-defined *modeling classes* and *relation classes* (Fig. 2). Modeling and relation classes may both have attributes. Functionality in ADOxx is inherited from the abstract pre-defined classes of the ADOxx meta-metamodel. The ADOxx metamodel is decomposed into two parts, a static part that features pre-defined abstract classes to represent organizational structures like, departments, actors, and roles, and a dynamic part (visualized in Fig. 3) that features pre-defined abstract classes that enable the realization of graph-based process-like metamodels.

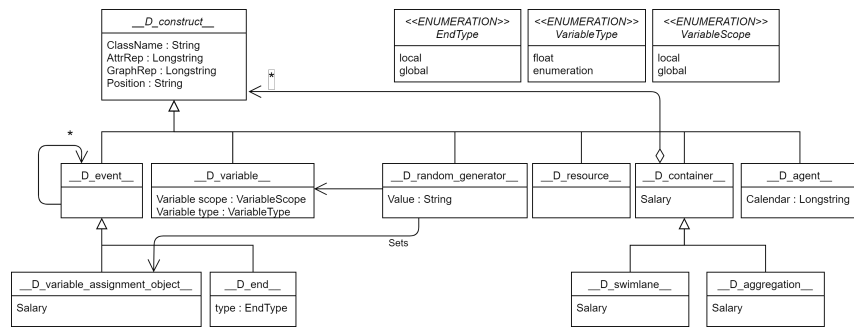


Fig. 3: Excerpt of the ADOxx metamodel

## 2.3 EMF

The Eclipse Modeling Framework (EMF) provides a meta-metamodel called Ecore<sup>3</sup> which can be considered as the Java-based realization of the Meta-Object Facility (MOF) standard [8]. With Ecore, language developers can define their own modeling languages in terms of metamodels and use different code generators to produce model APIs to

<sup>3</sup> <https://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html>



subject in [28]. UML profiles are derived to exchange models from EMF to UML and back again with the help of model transformations.

Going beyond concrete metamodeling platforms was first presented and discussed by Bezivin et al. [3]. Their main idea is to introduce technical spaces which provide a dedicated and representative meta-metamodel for a technical space as well as associated meta-modeling architecture and associated tools. This seminal work gave rise to a bunch of work on bridging several concrete technical spaces. When it comes to the modelware technical space, which is the technical space of metamodeling platforms, concrete approaches for providing bridges with Grammarware [29], XMLware [23], JSONware [11], and OntoWare [24] have been presented.

An approach that establish a new meta-layer on top of several existing metamodels in the business process modeling domain is presented by Heidari et al. [15]. The authors generalize and integrate the M2-level concepts of seven business process modeling language by proposing a generic M3-level domain metamodel.

From the perspective of ADOxx, several integrations with third-party tools (e.g., [10]) and concepts for metamodel patterns [21] and method chunks [25] as foundations for integrating heterogeneous metamodels have been proposed. To the best of our knowledge, an integration between ADOxx and other metamodeling platforms has not been discussed or realized in the past.

**Synopsis** To the best of our knowledge, there is a lack of approaches which consider the establishment of a bridge between ADOxx and EMF for exchanging metamodels and models between these two popular metamodeling platforms. However, previous interoperability architectures exploiting metamodeling stacks in combination with model transformations is a general solution scheme which is also employed in this paper. Nevertheless, the focus of this work is on the concrete interoperability challenges between the two modeling platforms as is discussed next.

## 4 Comparative Analysis of ADOxx and EMF Meta-Metamodels

In this paper, we are focusing on the abstract syntax definition exchange between both platforms. Thus, we only consider ADOxx and EMF without additional plug-ins and extensions – we leave these investigations as subject for future work. The comparative analysis in the following thus focuses on a thorough investigation of the meta-metamodels of the two platforms and their underlying metamodeling concepts. As can already be grasped from the introduction of the platform foundations in Section 2, ADOxx and EMF share many similarities while they also differ in some details. We now focus our attention to the core of the two platforms, their meta-metamodels and the metamodeling concepts they employ. The thorough analysis that follows is inspired by previous works (cf., e.g., [17]) and the experiences of the authors in realizing modeling tools with ADOxx and EMF.

Table 1 summarizes the findings of our analysis, differentiated along several categories. The table shows, whether and how a particular criteria is supported by the two platforms. In the following, some of the most interesting differences will be highlighted as they establish the major challenges for realizing interoperability – the concepts we developed to address these challenges are discussed in detail in Section 5.

Table 1: Comparison of M3 Level features of ADOxx and Ecore meta-metamodels

Criteria	ADOxx	Ecore
<b>Core Modeling Concepts</b>		
Class	Class	EClass
Relationship	Relation Class	EReference
Attribute	Attribute/Class Attribute	EAttribute
<b>Classes</b>		
Abstract Classes	✓	✓
User-defined root element	✗ <sup>1</sup>	✓
<b>Relationships</b>		
Arity	binary <sup>2</sup>	binary
Inverse	✗ <sup>3</sup>	✓
Composition	✗ <sup>3</sup> (only visual)	✓
Multiplicity	✓	✓
Endpoints	Class	EClass
Unique Names	✓ (per Metamodel)	✓ (per Class)
Link to Model	✓	✗
<b>Attributes</b>		
Applicable to	Class/Relation Class	EClass
Multiplicity	single-/multi-valued	single-/multi-valued
Unique	✓	✓
Ordered	✗ <sup>3</sup>	✓
Default Value	✓	✓
Custom Data Type	✓ <sup>4</sup>	✓
<b>Inheritance</b>		
Single/Multiple	single	multiple
Instantiation	single	single
Class Inheritance	✓	✓
Relationship Inheritance	✗	✗
<b>Grouping</b>		
	ModelTypes	EPackage

<sup>1</sup> every class in ADOxx inherits from a predefined abstract class<sup>2</sup> n-ary with *Interref*<sup>3</sup> realization via AdoScript possible<sup>4</sup> via Record Classes

**Composition** In EMF metamodels, composition plays an essential role. Any EMF metamodel needs to have a user-defined root class that contains directly or transitively any other metamodel class. On the ADOxx side, composition is not natively supported. ADOxx features an abstract pre-defined class *\_D\_container\_* with additional abstract subclasses (see Fig. 3). These abstract classes feature an automated detection mechanism that recognizes objects that are located geographically *inside* a *\_D\_container\_* object. Additional behavior such as cascaded deletion of composite objects is not natively supported.

**Single- vs. multiple inheritance** EMF supports multiple inheritance between classes whereas ADOxx supports single inheritance.

**Relation Class Uniqueness** Relation classes names in ADOxx are unique whereas for Ecore metamodels, no uniqueness of EReferences names is required as they are contained by the classes, thus having their own name space.

**Pre-defined Metamodel** The ADOxx metamodel comprises both, the abstract pre-defined classes and the user-defined classes whereas Ecore metamodels are solely composed of the user-defined classes, i.e., direct instantiations from M3.

## 5 Metamodel Transformation Approach

For operationalizing the mapping specified in Table 1, we implemented a bi-directional transformation chain. In particular, to achieve bidirectional transformations between ADOxx and EMF metamodels, we realized two unidirectional transformations (see Fig. 5), one transforming an ADOxx metamodel into an equivalent EMF metamodel and one the other way around. Note, that due to specific heterogeneities of the two platforms, roundtrip transformations between ADOxx and EMF will not result in an identical metamodel compared to the initial metamodel the roundtrip started from (see a discussion of selected heterogeneities in the previous section). Our approach therefore aims to achieve equivalence between the source and the target (i.e., transformed) metamodel when being used in the source and target metamodeling platforms, respectively. A detailed discussion on this matter is part of our evaluation in Section 6.

### 5.1 From ADOxx to EMF metamodels

For transforming ADOxx metamodels into EMF, we first use the XML export functionality provided by ADOxx. The derived XML-based metamodel specification is then parsed and processed in Java using JAXB. Eventually, we use the EMF API to create an equivalent Ecore metamodel and to serialize it into standardized XMI format which enables direct loading into EMF.

**Pre-defined metamodel** As mentioned earlier already, ADOxx metamodels are composed of abstract pre-defined classes and user-defined classes. In order to not obscure the EMF metamodel and to enable focus on the user-defined metamodel, we separated the two metamodels into two *EPackages*.

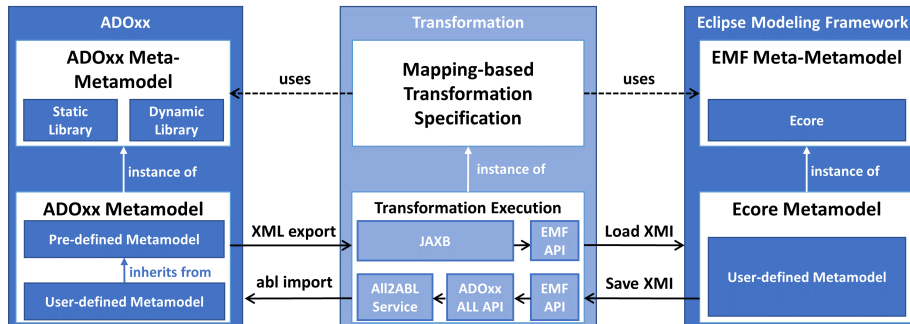


Fig. 5: Technological view on the two unidirectional transformations

**User-defined root class** As any Ecore metamodel needs to have a single, user-defined root class that has no counterpart in ADOxx metamodels, we generate a root class with the name of the ADOxx library the transformation was initiated with.

**Enumerations** Enumerations in ADOxx are treated as conventional attributes of modeling and relation classes whereas in Ecore-based metamodels, enumerations are special classes. Consequently, all enumerations in an Ecore-based metamodel also need to have unique names which is not ensured from the ADOxx side, as enumerations attributes only need to have a unique name per class, not per library. We solve this challenge by prefixing the name of a transformed enumerations class in Ecore with the name of the ADOxx class the enumerations attribute originally belongs to.

**Java Identifiers** ADOxx allows to use various characters in class and attribute names (e.g., dots and spaces), which are not allowed in Ecore where all names need to follow the naming rules of Java identifiers. We solved this issue by replacing all prohibited characters by either an underscore, or omitting them all together.

## 5.2 From EMF to ADOxx metamodels

For transforming EMF metamodels into ADOxx, we first use the standardized XMI serialization. We then use the Java EMF API to process the Ecore-based metamodel and apply the respective transformation rules. To generate the equivalent ADOxx metamodel, we use the *ADOxx ALL API*<sup>4</sup>. Once the generation is concluded, the *ALL2ABL web service*<sup>5</sup> transforms the textual metamodel specification in ALL into an ADOxx application library (.abl) file that can be imported into ADOxx.

**Composition** To simulate composition in ADOxx, we generate two AdoScript event handlers for each composition in Ecore. One event handler is executed when a modeler triggered the creation of a new instance of a modeling class (i.e., *AfterCreateModelingNode* event in ADOxx) and one in cases where the modeler has triggered the deletion of a modeling instance (i.e., *BeforeDeleteInstance* event in ADOxx). Aside from the event handlers, the transformation creates a library attribute named *compositeClasses* that stores all class names of composite classes and, for each composite class in Ecore, an attribute *compositumClass* in the ADOxx metamodel. These attributes are used in the event handlers to navigate from composite to compositum and vice versa. Algorithm 1 describes these two event handlers in pseudo code. The first ensures, that each newly created composite needs to be linked to a valid compositum, whereas the second ensures that the deletion of a compositum is cascaded to all its composite objects.

**Multiple Inheritance** Several approaches have been proposed in the literature to translate multiple inheritance structures into equivalent single inheritance structures (cf. [12, 13]). After carefully studying potential solutions, we opted to apply an adapted version of the *expansion pattern* originally proposed by Crespo et al. [12]. Fig. 7 visualizes the pattern in detail. Our transformation picks the first superclass to inherit from, the properties of all remaining super classes are then reproduced

<sup>4</sup> <https://www.adoxx.org/live/adoxx-java>

<sup>5</sup> <https://www.adoxx.org/live/all2abl-converter-service>

**Algorithm 1:** AdoScript code for handling composition.

---

**Input:** *classid*, *objid*, and *modelid* of the object *o* to be created

```

1  ON_EVENT "AfterCreateModelingNode"
2  compositeClasses ← LibraryMetaData.compositeClasses()
3  if compositeClasses contains classid then
4      compositumClass ← o.compositumClass()
5      availableCompositumObjects ← GET_ALL_OBJS_OF_CLASSNAME(modelid, compositumClass)
6      if availableCompositumObjects.size() > 0 then
7          selectedCompositumObject ← LISTBOX(availableCompositumObjects).selection()
8          ADD_INTERREF(selectedCompositumObject, o)
9      else
10         DELETE_OBJ(o)

```

**Input:** *classid*, *objid*, and *modelid* of the object *o* to be deleted

```

11  ON_EVENT "BeforeDeleteInstance"
12  compositeClasses ← LibraryMetaData.compositeClasses()
13  compositumClasses ← LibraryMetaData.compositumClasses()
14  if compositumClasses contains classid then
15      for Composite c : o.composedInstances() do
16          DELETE_OBJ(c)
17      end
18  else if compositeClasses contains classid then
19      compositumClass ← o.compositumClass()
20      availableCompositumObjects ← GET_ALL_OBJS_OF_CLASSNAME(modelid, compositumClass)
21      for Compositum com : availableCompositumObjects do
22          if com.composedInstances().contains o then
23              REMOVE_INTERREF(com, o)
24      end

```

---

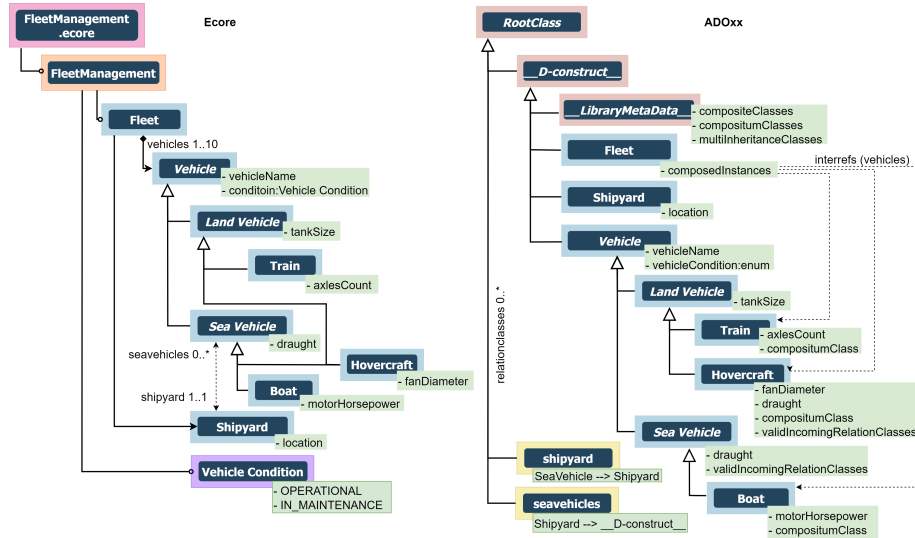


Fig. 6: Illustrative multiple inheritance transformation example: EMF to ADOxx

to ensure that the subclass (class *D* in Fig. 7) has the same expressiveness as when actually inheriting from multiple classes. Fig. 6 summarizes the explained mapping by example. Note that *Vehicle Condition* is an enumeration, *shipyard* and *seavehicles* are relationships.

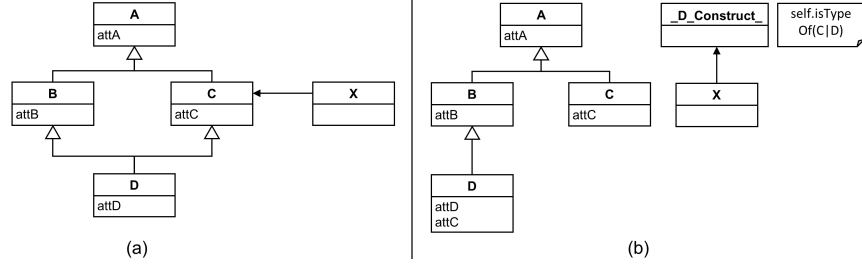


Fig. 7: Multi-inheritance example (a) and the adapted expansion pattern (b)

Additionally, we need to make sure to handle the incoming relationships of the remaining super classes (class C in Fig. 7) as these shall be also valid for the inheriting class D. We decided to solve this by changing the endpoint of the relation class from the specific class (in this case class C) to the abstract super class `D_Construct_` in ADOxx (i.e., now this relation can be incoming to any user-defined class). To restrict the possible endpoints to the valid ones, we further generate an event handler in AdoScript and create a LibraryMetaData `multiInheritanceClasses` attribute in ADOxx that stores the class names of all remaining classes and the inheriting class of multiple inheritance (i.e., the classes C and D in Fig. 7).

Algorithm 2 describes the event handler for realizing multiple inheritance in ADOxx. It is executed when the modeler triggered the creation of a new relation and checks, whether the class name of the *to object* of the newly created relation participates in multiple inheritance. Then, the algorithm checks, whether the relation class is valid to be connected with the *from object* by means of an incoming relation.

---

**Algorithm 2:** AdoScript code for handling multiple inheritance.

---

```

Input: classid, relationid, modelid, toObj, and fromObj of the relation r to be created
1 ON_EVENT "AfterCreateModelingConnector"
2 multiInheritanceClasses  $\leftarrow$  LibraryMetaData.multiInheritanceClasses()
3 if multiInheritanceClasses contains classid then
4   validIncomingRelationClasses  $\leftarrow$  toObj.validIncomingRelationClasses()
5   if !validIncomingRelationClasses contains classid then
6     DELETE_CONNECTOR(r)
7   end
8 end

```

---

**Relation class uniqueness** As EReferences only require unique names on class level and not in the entire Ecore metamodel, but Relation Class names in ADOxx are unique in the entire metamodel, we adjusted the transformation in such a way, that the name of the target Relation Class in ADOxx is composed as follows: `<source-classname><EReference-name><target-classname>`.

## 6 Evaluation

For evaluating our transformation chain, we conducted a set of experiments and applied different means of quantitatively and qualitatively testing the transformation.

### 6.1 Research Questions & Evaluation Methods

With the evaluation, we aim to respond to the following three major research questions.

**RQ1-Metamodel Validity:** *Does our transformation produce valid metamodels in the target metamodeling platform?* For testing the validity of the transformed metamodels, we use platform-specific built-in functionality. For validating Ecore metamodels, we first apply the *EMF Validator* and secondly register the metamodel. The ADOxx metamodels are automatically validated once an .abl file is imported.

**RQ2-Metamodel Expressivity:** *Is the expressivity of the source and target metamodel equivalent?* To test the expressivity equivalence, we imported the transformed metamodels and manually investigated their expressivity on both platforms. We further developed metrics on both platforms that enable the automated evaluation of the metamodel expressivity.

**RQ3-Model Processing Equivalence:** *Is the behavior of processing a model in the two platforms equivalent?* Here, we instantiated sample models using the source and the target metamodels in their respective platforms. We then applied CRUD operators to see, whether the models adhering to the transformed metamodel in the target platform behave equivalent to the behavior of the model adhering to the source metamodel in the source platform.

### 6.2 Study Setup

For the study we used 45 ADOxx metamodels, reported in [5], that were transformed into Ecore metamodels. Furthermore, we used 33 randomly selected Ecore metamodels of the ATL zoo<sup>6</sup>, to transform them into equivalent ADOxx metamodels. Table 2 provides some descriptive statistics on the metamodels we used in our evaluation experiments. It shows the minimal, median, and maximal number of classes, number of abstract classes, number of classes with multiple inheritance, relationships, attributes, enumerations, and the maximal inheritance depth of the metamodels.

### 6.3 Results

**RQ1-Metamodel Validity.** None of the generated metamodels yield errors or warnings when being imported into EMF (see Table 3). The implementation of semantically appropriate procedures for renaming classes and attributes in order to comply to naming conventions, separating various elements into different packages in order to avoid name clashes, have thus been validated. Initial tests pointed us to these issues. Another issue was resolved by ensuring that attributes with the same name are neither defined in the same class nor in one of its super classes.

One remaining issue we observed relates to the data types. In one instance, a certain value of an attribute is too long for the datatype STRING, which is limited to a max length of 3699 symbols in ADOxx. This error occurs in the *AttributeInterRefDomain* facet, which has a non-changeable datatype of STRING. We set this facet for defining all composed instances of a compositum class in order to enable composition

<sup>6</sup> <https://www.eclipse.org/atl/atlTransformations/>

on ADOxx side. For one particular library of our experiments, so many composed instances exist, that the generated string is simply too long for this datatype. We already have a workaround in mind but need to test it thoroughly in future experiments.

**RQ2-Metamodel Expressivity.** We have analyzed the expressivity of a representative sample of the transformed metamodels by analyzing their structure and their created classes, attributes, and relationships. We found that the mapping of the different elements corresponds to our specification. In some cases, such as in the ADOxx to EMF transformation, we verified that the correct amount of additional Enum-Classes was created and correctly assigned to the appropriate metamodel class. In other cases, we verified that relations are correctly assigned from the source to the target class, as well as their cardinalities are set correctly. Further, in the EMF to ADOxx transformation, we verified that the correct metadata for multiple inheritance and composition is set in the correct classes on ADOxx side.

**RQ3-Model Processing Equivalence.** As a final evaluation step, we have created sample models by using the generated metamodels. Here, we could verify that the class creation, attribute value modification, composition features, and multiple inheritance features behave equivalent in both platforms. For the composition part on the ADOxx side, we verified that the transformed event handlers work correctly. For this, we generated compositum and composite elements, then deleted the compositum element, and verified, that all composite instances are deleted as well. For the multiple inheritance transformation on ADOxx side, we checked, that for any class that inherits multiple classes in the Ecore metamodel, the additional attributes from the non-inherited super class are present in the corresponding ADOxx class instance. Additionally, we verified the correct behaviour of our event handler in allowing also incoming relationships from the other multi-inheritance super classes.

**Limitations.** Our study of course also comes with limitations, some of which will be briefly discussed in the following. First, we need to limit the generalizability of our outcomes. We cannot generalize our results to other metamodels having different structures, sizes, etc. Our experiments incorporated 33 EMF and 45 ADOxx metamodel-

Table 2: Metrics of the source ADOxx and EMF metamodels used in the experiments

	ADOxx			EMF		
	Min	Med	Max	Min	Med	Max
Classes	5	30	180	1	7	93
Abstract Classes	0	2	24	0	1	12
Relations	1	11	81	0	2	59
Compositions	0	2	13	0	2	36
Attributes	2	86	1165	1	6	64
Inheritance Depth	1	3	6	0	1	4
Multi Inheritance Classes	-	-	-	0	0	4
Enumerations	0	17	270	0	0	7

Table 3: Outcomes of the conducted metamodel transformation experiments

Transformation Direction	Cases	No errors	Error	Success rate
ADOxx $\rightarrow$ Ecore	45	45	0	100 %
Ecore $\rightarrow$ ADOxx	33	32	1	96.97 %

els of different nature (see Tab. 2 for some statistics), still more experiments need to be conducted to further generalize our findings. Second, our transformations may not be representative or amenable for language engineers who may have other patterns in mind to represent the metamodels in the other platform. We took deliberate decisions, based on the literature and also the experience of the authors working with both platforms, when designing the transformations, esp. for the challenging parts reported on in Section 5, still others may prefer to, e.g., follow a different multi-to-single inheritance transformation pattern, etc. Third, some metamodel functionalities provided by ADOxx (e.g., *conversion*) and EMF (e.g., *eOpposite* and *move*) are not considered yet in the transformation as we focus on information loss exchange between the platforms. Future research will investigate how to consider such features in the transformation.

**Future Research.** In order to mitigate some of the limitations, a detailed analysis at the transformed source and target metamodels will be conducted. Work on this task has started, where we compare different features such as class size, relation size, enumeration size and so forth for each individual source and generated target metamodel. These metrics provide a better understanding of the nature of the different metamodels and were inspired from previous work [5, 14, 19, 22]. A comparison on the individual metamodel-level will reveal the concrete impacts of a transformation, e.g., how many attributes have been created in the target metamodel from a set of attributes in the source metamodel. General findings about the complexity of the generated metamodels and the performance of the transformation can be found this way.

On the other hand, this procedure can support the validation of the transformation by comparing the different metrics in source and target metamodel to derive conclusions. For instance, since our algorithm transforms Interrefs from an ADOxx source metamodel to relationships in Ecore, we may verify that the total amount of expected relationships in Ecore can be calculated as follows: Given the ADOxx relationships count as  $|R_{ado}|$  and the ADOxx Interrefs count  $|IR_{ado}|$ , the total amount of generated relationships in the target Ecore metamodel  $|R_{emf}|$  would need to be  $|R_{emf}| = |R_{ado}| + |IR_{ado}|$ .

## 7 Conclusion

In this paper, we have presented a transformation-based framework to exchange metamodels between ADOxx and EMF. We evaluated the resulting bridge by transforming a representative set of metamodels and evaluated the outcomes with respect to validity, expressivity, and modeling equivalence. From a scientific point of view, this work investigated the different concepts provided by ADOxx and EMF. Although, there are several differences, we have also shown that there is a common core which can be used to simulate the missing parts in both worlds. From a practical point of view, this work

supports the exchange of metamodels between both platforms which allows for reuse of existing metamodels available only for one platform as well as using a wide variety of tools for the metamodels which are available in both platforms. The bridge is deployed openly and can be used via: <http://me.big.tuwien.ac.at/>.

The evaluation conforms already a very high level of interoperability enabled by our bridge. The single remaining error is minor and we already have ideas for solving it. Future research will focus on mitigating this remaining issue, drilling down other edge cases that we did not yet encounter, and extending the transformation in two ways. First, we are currently investigating the extent to which the graphical concrete syntax of the metamodel concepts can be transformed between the two platforms. Second, we aim to extend the transformation to be also applicable to models instantiated from the transformed metamodels to allow not only metamodel exchange between the platforms, but model exchange as well.

*Acknowledgements.* Work partially funded by the Austrian Science Fund (P 30525-N31) and by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development (CDG).

## References

1. ADOxx.org: Official homepage of the ADOxx meta-modeling platform (2021), <http://adoxx.org>, last visited: 27.03.2022
2. Bézivin, J., Brunette, C., Chevrel, R., Jouault, F., Kurtev, I.: Bridging the Generic Modeling Environment (GME) and the Eclipse Modeling Framework (EMF). In: Proceedings of the Best Practices for Model Driven Software Development at OOPSLA (2005)
3. Bézivin, J., Devedzic, V., Djuric, D., Favreau, J.M., Gasevic, D., Jouault, F.: An M3-Neutral infrastructure for bridging model engineering and ontology engineering. In: Interoperability of enterprise software and applications, pp. 159–171. Springer (2006)
4. Bider, I., Perjons, E., Bork, D.: Towards on-the-fly creation of modeling language jargons. In: Proceedings of the 17th International Conference on ICT in Education, Research and Industrial Applications. CEUR, vol. 3013, pp. 142–157. CEUR-WS.org (2021)
5. Bork, D.: Metamodel-based Analysis of Domain-specific Conceptual Modeling Methods. In: Buchmann, R.A., Karagiannis, D., Kirikova, M. (eds.) IFIP Working Conference on The Practice of Enterprise Modeling. pp. 172–187. Springer (2018)
6. Bork, D., Buchmann, R., Hawryszkiewicz, I., Karagiannis, D., Tantouris, N., Walch, M.: Using conceptual modeling to support innovation challenges in smart cities. In: IEEE 14th International Conference on Smart City. pp. 1317–1324 (2016)
7. Bork, D., Buchmann, R.A., Karagiannis, D., Lee, M., Miron, E.T.: An Open Platform for Modeling Method Conceptualization: The OMiLAB Digital Ecosystem. Communications of the Association for Information Systems **44**, pp. 673–697 (2019)
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, Second Edition. Morgan & Claypool Publishers (2017)
9. Brunelière, H., Cabot, J., Clasen, C., Jouault, F., Bézivin, J.: Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools. In: Modelling Foundations and Applications - 6th European Conference, ECMFA 2010, Proceedings. pp. 32–47. Springer (2010)

10. Buchmann, R.A., Karagiannis, D.: Domain-specific diagrammatic modelling: a source of machine-readable semantics for the Internet of Things. *Clust. Comput.* **20**(1), 895–908 (2017)
11. Colantoni, A., Garmendia, A., Berardinelli, L., Wimmer, M., Bräuer, J.: Leveraging model-driven technologies for JSON artefacts: The shipyard case study. In: 24th International Conference on Model Driven Engineering Languages and Systems (MODELS). pp. 250–260. IEEE (2021)
12. Crespo, Y., Marques, J.M., Rodríguez, J.J.: On the translation of multiple inheritance hierarchies into single inheritance hierarchies. In: Proceedings of the Inheritance Workshop at ECOOP, pp. 30–37 (2002)
13. Dao, M., Huchard, M., Rouge, T.L., Pons, A., Villerd, J.: Proposals for multiple to single inheritance transformation. In: MASPEGHI: Managing SPecialization/Generalization Hierarchies. pp. 21–26. Laboratoire I3S (Rapport de recherche) (2004)
14. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining metrics for understanding metamodel characteristics. In: Proceedings of the 6th International Workshop on Modeling in Software Engineering (MiSE). ACM (2014)
15. Heidari, F., Loucopoulos, P., Brazier, F.M.T., Barjis, J.: A meta-meta-model for seven business process modeling languages. In: IEEE 15th Conference on Business Informatics (CBI). pp. 216–221. IEEE (2013)
16. Kelly, S., Lyytinen, K., Rossi, M.: Metaedit+: A fully configurable multi-user and multi-tool CASE and CAME environment. In: Advances Information System Engineering, 8th International Conference, CAISE'96, Proceedings. pp. 1–21. Springer (1996)
17. Kern, H.: The Interchange of (Meta)Models between MetaEdit+ and Eclipse EMF Using M3-Level-Based Bridges. In: Tolvanen, J., Gray, J., Rossi, M., Sprinkle, J. (eds.) 8th Workshop on Domain-Specific Modeling at OOPSLA. ACM (2008)
18. Kern, H.: Model interoperability between meta-modeling environments by using M3-level-based bridges. Ph.D. thesis, Leipzig University, Germany (2016)
19. Kern, H., Hummel, A., Kühne, S.: Towards a comparative analysis of meta-metamodels. In: 11th Workshop on Domain-Specific Modeling at OOPSLA. pp. 7–12. ACM (2011)
20. Kern, H., Kühne, S.: Model Interchange between ARIS and Eclipse EMF. In: 7th Workshop on Domain-Specific Modeling at OOPSLA. vol. 2007 (2007)
21. Kühn, H., Bayer, F., Junginger, S., Karagiannis, D.: Enterprise model integration. In: Bauknecht, K., Tjoa, A.M., Quirchmayr, G. (eds.) E-Commerce and Web Technologies, 4th International Conference, EC-Web. pp. 379–392. Springer (2003)
22. Langer, P., Mayerhofer, T., Wimmer, M., Kappel, G.: On the Usage of UML. In: Fill, H.G., Karagiannis, D., Reimer, U. (eds.) Modellierung 2014. pp. 289–304. GI (2014)
23. Neubauer, P., Bergmayr, A., Mayerhofer, T., Troya, J., Wimmer, M.: XMLText: from XML Schema to Xtext. In: Proc. of the ACM SIGPLAN International Conference on Software Language Engineering (SLE). pp. 71–76. ACM (2015)
24. Parreiras, F.S., Staab, S.: Using ontologies with UML class-based modeling: The TwoUse approach. *Data Knowl. Eng.* **69**(11), 1194–1207 (2010)
25. Ralyté, J., Backlund, P., Kühn, H., Jeusfeld, M.A.: Method chunks for interoperability. In: 25th International Conference on Conceptual Modeling. pp. 339–353. Springer (2006)
26. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Addison-Wesley (2008)
27. Tratt, L.: Model transformations and tool integration. *Softw. Syst. Model.* **4**(2), 112–122 (2005)
28. Wimmer, M.: A semi-automatic approach for bridging DSMLs with UML. *Int. J. Web Inf. Syst.* **5**(3), 372–404 (2009)
29. Wimmer, M., Kramler, G.: Bridging grammarware and modelware. In: Bruel, J. (ed.) Satellite Events at the MoDELS 2005 Conference. pp. 159–168. Springer (2005)