Establishing Interoperability between the EMF and the MSDKVS Metamodeling Platforms

Florian Cesal and Dominik Bork

To appear in:

15th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling (PoEM'2022)

Final version available soon:

www.model-engineering.info

Establishing Interoperability between the EMF and the MSDKVS Metamodeling Platforms

Florian Cesal and Dominik Bork[0000−0001−8259−2297]

TU Wien, Business Informatics Group, Favoritenstrasse 9-11, 1040 Vienna, Austria
dominik.bork@tuwien.ac.at

Abstract. Many powerful metamodeling platforms exist, each with strengths, weaknesses, functionalities, programming language(s), and developer community. To exploit the mutual benefits of these platforms, it would be ideal to establish interoperability amongst them and the exchange of metamodels and models. This would enable language engineers to choose the metamodeling platform freely without risking a lock-in effect. Two well-documented and freely available metamodeling platforms are the Eclipse Modeling Framework (EMF) and Microsoft's Modeling SDK for Visual Studio (MSDKVS). This paper proposes the first achievements toward establishing interoperability between EMF and MSDKVS on an abstract syntax level and a graphical concrete syntax level. To develop such interoperability, we i) comprehensively analyze the two platforms, ii) present a conceptual mapping between them, and iii) eventually implement a bidirectional transformation bridge. The transformed results' validity, executability, and expressiveness are then quantitatively and qualitatively assessed by transforming a collection of publicly available metamodels.

Keywords: Metamodeling · Interoperability · EMF · Sirius · DSL · MSDKVS.

1 Introduction

The definition and use of modeling languages offer many benefits in how software teams and language designers can efficiently cooperate on creating a model-based representation of the system under study. Metamodeling platforms offer means to define customized languages easily and many additional functionalities such as code generation, automatic validation, and graphically representing models. These platforms are widely used in enterprise modeling and model-driven software engineering (MDSE). However, once modellers work with one platform, switching to a different one is cumbersome, complex, and costly, especially because automated support for metamodeling platform interoperability is scarce.

This paper looks at the two well established and actively used metamodeling platforms Eclipse Modeling Framework (EMF) [23] and Modeling SDK for Visual Studio (MSDKVS) [22]. We propose a transformation bridge between EMF and MSDKVS related to bridges reported in [3, 15, 16, 18]. Such a bridging enables language designers to seamlessly switch between platforms by transforming already defined metamodels in one platform into syntactically and semantically equivalent metamodels in the target platform. These bridges further enable reusability of existing metamodels

in other platforms, decouple the developers of the underlying programming languages these platforms are built upon, and enable making use of specific platform capabilities employed elsewhere. The transformation bridges are based on a mapping between the meta-metamodel concepts of both platforms. This mapping is created by analyzing the similarities and identifying the differences between these platforms located at the M3 layer of the standardized metamodeling stack [6]. The previous approaches implemented transformation bridges targeting the platforms' abstract syntax elements (e.g., classes and relationships), mostly ignoring the platform's functionalities to graphically represent and manipulate the created models. This paper first extensively analyses the EMF and MSDKVS platforms and then proposes, implements, and evaluates a transformation bridge, thereby considerably advancing previous attempts (see Section 3.2).

In the remainder of this paper, Section 2 establishes the foundations. Section 3 then discusses related works. A comprehensive analysis of the platforms' concepts is presented in Section 4 which establishes the foundation for the bridging in Section 5. Section 6 then evaluates the bridge before conclusions are discussed in Section 7.

2 Metamodeling Foundations

Complete or partial representations of real-world objects, architectures, or software systems can be realized through the use of models. These models can then be shared and enable communication among stakeholders [6]. Concerning the validation and guidelines for defining said models, an abstraction hierarchy exists, divided into a stack of layers. An example of such a hierarchy stack, consisting of four layers, has been standardized by the OMG [6, 19]: **M0 Layer (runtime instances)** containing the application data or runtime instances; **M1 Layer (model layer)** describing the concrete user model based on the given metamodel (e.g. a UML model); **M2 Layer (metamodel layer)** defining the metamodel; **M3 Layer (meta-metamodel layer)** abstracting the definition for possible metamodels. In the OMG metamodeling hierarchy, the M3 layer is defined by the Meta-Object Facility (MOF) standard.

The M3 level also establishes the foundation for realizing interoperability of metamodeling platforms based on a common abstraction of their metamodels. Modeling languages consist of the following elements, which are taken into consideration when implementing the transformation bridge in the subsequent sections: **Abstract Syntax:** defines classes, their attributes, and associations required to represent the relevant parts of the modeled system, and constraints for restricting the set of valid models. Abstract syntaxes are most often specified via metamodels [5]. **Concrete Syntax:** defines the visual appearances for the abstract syntax elements (e.g., graphical and/or textual) [4].

In practice, language engineers and tool developers need to decide on the most appropriate metamodeling platform considering the requirements at hand. Metamodeling platforms provide IDEs to efficiently create the abstract and concrete syntax of modeling languages, generate editors to create models based on defined metamodels, define and execute code generators and model transformations. Two prominent exemplars of such platforms are introduced in the following.

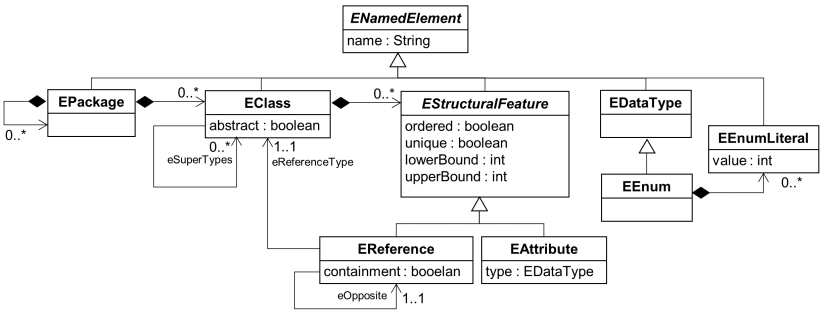


Fig. 1: Excerpt of the Ecore meta-metamodel [3]

2.1 EMF

The Eclipse Modeling Framework (EMF) is an open source metamodeling platform that provides a rich set of features for, e.g., defining metamodels, creating and validating models, transforming models, and serializing models into XMI format. EMF also allows runtime support to generate Java classes and programmatically manipulate the models through reflection. This section describes some of the core features of EMF and provides an overview of how metamodeling is realized with EMF.

Abstract Syntax in EMF. To realize metamodel support in EMF one needs to specify the metamodel by instantiating concepts from the EMF meta-metamodel. This meta-metamodel (see Fig. 1 for an excerpt) thus plays an essential role as it determines the expressiveness of all possible metamodels. An explicit definition of the Ecore meta-metamodel is given in various sources, e.g., in [3, 6, 18].

Concrete Syntax in EMF. The Eclipse website lists three frameworks that can be used for visualizing Ecore metamodels and models (i.e., Graphical Language Server Platform, Sirius, and Graphiti). For the matter of this paper, we only consider Sirius as it is the most commonly used framework and best resembles the possibilities of graphical viewpoint representations on the MSDKVS side. Sirius uses so called *Viewpoint Specification Projects (VSP)* containing descriptive model files ending with *.odesign* [24]. These files contain the specification of the graphical representation of a model and are comprised of layer definitions and tool sections containing toolbox operations with a structured dependency tree of further inner operation mappings, style mappings for model shapes, font layout properties, custom color definitions, and much more.

2.2 MSDKVS

MSDKVS supports the development of domain-specific languages by weaving abstract syntax and graphical concrete syntax (see [9, 22] for a detailed introduction). MSDKVS offers a graphical user interface with an integrated editor to define metamodels (i.e., classes, relationships, and their properties) and a tree explorer, a property editor window, and several other features such as XML serialization of metamodels and models, code generators using a templating engine, and the possibility to build extensions to these features. The currently available MSDKVS NuGet Package[1] was released six months ago and still has an active community of language designers.

[1] https://www.nuget.org/packages/Microsoft.VisualStudio.Modeling.Sdk

Abstract Syntax in MSDKVS. As MSDKVS does not publicly offer a view on the meta-metamodel, the transformation approach explained in Section 5 implicitly offers the ability to generate a MSDKVS meta-metamodel corresponding to the data of the serialized metamodel files. Fig. 2 shows the core excerpt of the generalized version of the MSDKVS meta-metamodel as a UML class diagram, the full representation is provided here[2]. When creating a DSL in MSDKVS, one element always has to act as the root element of the metamodel and every subsequently created *DomainClass*, if not targeted by another embedded relationship, is referenced by this root class. The root class, by default, has the same name as the DSL itself, but can be changed after initial creation. The possible entities that can be created on the metamodeling canvas are available in the Dsl Designer Toolbox. These elements include *DomainClasses* and different types of *DomainRelationships*, like embedding relationships (i.e., containers) and reference relationships. Each element can further be attributed with various *DomainProperties*.

Regarding their XML serialization, the DslDefinition.dsl file, when opened in a text editor, contains all objects added on the DSL canvas and their mapping references to tool palettes, shapes (i.e., graphical concrete syntax), and other serialization properties needed for code generation. Every object on the canvas is given a *Moniker* description type to be able to be referenced in other parts of the DSL. Monikers are uniquely identifying names for elements. The XML content of the metamodel is grouped in different functional areas where only the entities' moniker types are used as references, e.g., *Source* and *Target* role types of a *DomainRelationship*.

Concrete Syntax in MSDKVS. Every class, relationship, and attribute can be visually enhanced with different shapes and decorators that are maintained within the editor's graphical interface adjoined to the abstract syntax definitions. Through mapping links between the concrete and abstract syntax definitions the language designer can customize the appearances and interaction possibilities like toolbox entries or graphically editing attributes in the Visual Studio runtime instances. MSDKVS does not offer the possibility to define concrete syntax on the metamodel layer.

3 Related Work

This section first offers an overview of existing works on metamodeling platform interoperability. It then takes a detailed look at related ambitions toward bridging EMF and MSDKVS and compares these works to this paper's approach.

3.1 Transformation bridges

Interoperability deals with the exchange of information between two or more systems, and the ability to use that information in each system respectively [12]. As modeling languages for software development gained popularity in the early 2000s, a need for transforming the grammarware technical space (i.e., EBNF-based grammar tools) to the modelware technical space existed [25] to achieve interoperability between these sets of tools. When this interoperability was established, a plethora of metamodeling platforms followed, which in turn also raised the need for interoperability amongst them.

[2]Online supplementary material: `https://tinyurl.com/EMF-MSDKVS`

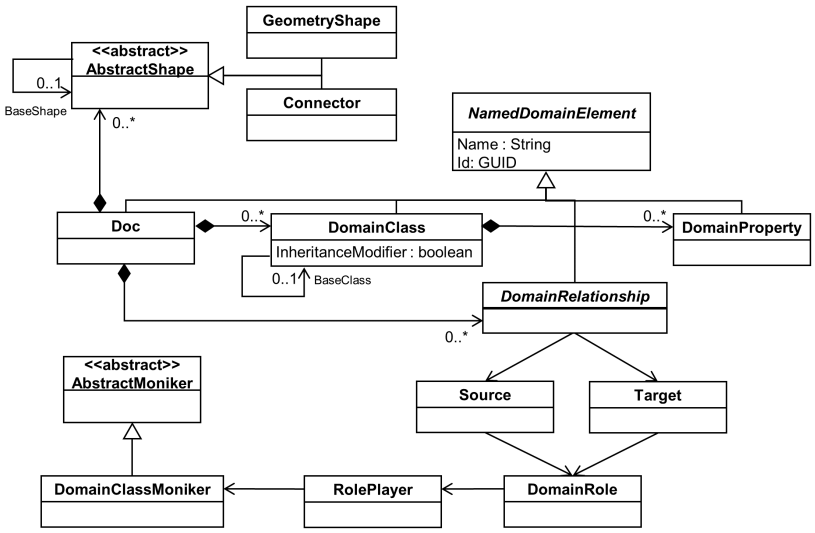


Fig. 2: Excerpt of the MSDKVS meta-metamodel

In the past several bridges between different metamodeling platforms have been proposed, including EMF and ARIS [16], EMF and MetaEdit+ [15], EMF and Visio [18], and EMF and Generic Modeling Environment [8]. Most recently, a transformation bridge between ADOxx and EMF has been proposed in [3]. These transformation bridges typically consist of one or several model transformations that are used for exchanging metamodels and models between the two platforms. These are so-called *horizontal exogenous transformations* [6, 13], as the source and target of the transformation are situated on the same abstraction level but adhere to different meta-metamodels. Most of these works transform metamodels, i.e., do not consider interoperability at the model level which further requires a transformation between concrete syntaxes.

3.2 EMF and Microsoft DSL Tools

Research on bridging EMF and Microsoft DSL Tools has been proposed in the past [1, 7]. Differences regarding today's version of MSDKVS as opposed to the transformation approach in [7] are e.g., the serialized file formats (.dsldm compared to today's .dsl mentioned in [1]), the visualization of a meta-metamodel containing the *ValueProperty* entity compared to today's *DomainProperty*, and the representation of attributes for classes and relationships.

The previous approaches execute a chain of ATLAS transformation language transformations to generate the transformed metamodel using the *KM3 (Kernel MetaMetaModel)*, a DSL for describing metamodels [14] as an intermediate representation of arbitrary metamodels. As a transformation already existed between KM3 and Ecore, the MS/DSL metamodels needed to be only transformed to this pivot KM3 metamodel. Thus, no direct transformation between EMF and MS/DSL tools existed, which introduces potential information loss as KM3 can be considered a generic platform-agnostic DSL to represent the 'common denominator' of several metamodels. We examined the

previous approach with preserved .dsldm files of the Atlantic-Zoo Github[3] and learned that the execution of the XML2DSL step always resulted in empty files. This is caused by the evolution of the MSDKVS platform (mentioned above) and the discontinuation of some of the used components in the previous approach.

This paper gives a detailed comparative analysis (see Section 4) of the abstract and concrete syntax elements available in the latest versions of the EMF and MSDKVS platforms that far exceeds previous works. Moreover, with this paper, we present the first direct transformation bridge that also transforms the graphical concrete syntax. One example transformation was explained in [7], the PetriNet metamodel, where the question remains if the validation of said transformed metamodel and models was successful in the target platform. In the paper at hand we address this gap by providing an exhaustive quantitative and qualitative evaluation of the transformation bridge (see Section 6).

4 Comparative Analysis of EMF and MSDKVS

This section comprehensively analyzes the two platforms regarding their abstract and concrete syntax capabilities. The list of relevant elements was adapted and extended from [3, 17] in terms of first-class concrete syntax concepts extracted through a detailed investigation of the platforms in question. A full list of the identified, analyzed, and mapped abstract and concrete syntax elements is provided in the online supplementary material[2]. In the following, we therefore concentrate the analysis on the core differences between EMF and MSDKVS as these differences establish the challenges and inform the design of a direct transformation bridge (see Section 5).

4.1 Abstract Syntax Features

EMF allows the definition of classes that inherit properties and possible relationship structures from multiple classes (i.e., **multiple inheritance**) whereas MSDKVS only allows entities to inherit from one referenced base object (i.e., **single inheritance**). MSDKVS, in contrast to EMF, allows **inheritance between relationships**. Furthermore, domain relationships in MSDKVS can also act as domain classes which can then be connected to other domain relationships as source or target role. One minor but challenging difference relates to the possibility of relationships between elements to have the **same name** in EMF, which leads to name clashes on the MSDKVS side where relationship names are required to be **unique**. On MSDKVS, domain classes are not directly referenced when creating a relationship. Instead, they are indirectly referenced through monikers, and a domain relationship is comprised of source and target **domain roles** also referencing these monikers. When implementing a transformation between MSDKVS and EMF, correctly resolving these indirect dependencies to achieve syntactical and semantical equivalence is very challenging.

[3] https://github.com/atlanmod/atlantic-zoo/tree/main/AtlanticDSLTools

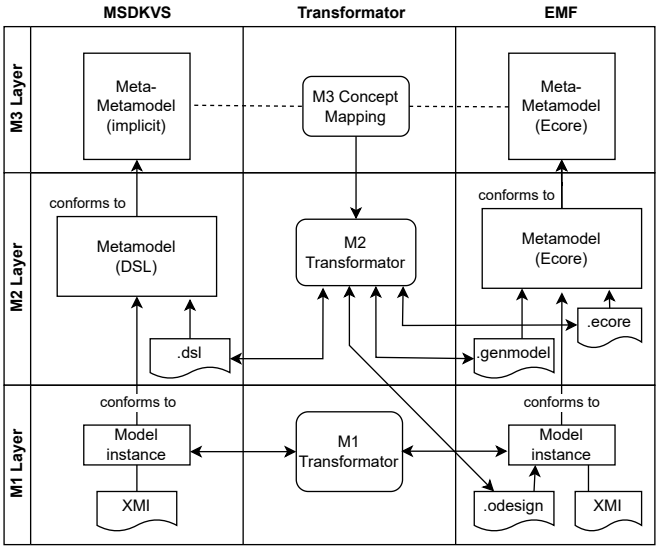


Fig. 3: Transformation bridge between EMF and MSDKVS

4.2 Graphical Concrete Syntax Features

MSDKVS offers the possibility to inherit properties among shapes (i.e., **shape inheritance**) while in EMF no inheritance between the graphical syntax specification is possible. Besides the support for widely used basic shapes like rectangles, circles, and icons, each platform offers *special shape types* that cannot be directly mapped to an equivalent one in the other platform. As metamodeling platforms often depend on an underlying programming language (e.g. EMF on Java, MSDKVS on C#), the available **coloring options**, **styles**, and **appearance attributes** are limited by the languages' libraries. As for MSDKVS, three different types of color palettes are available (system, web, and custom). EMF offers a selection of basic system colors per default. Metamodeling platforms also allow the use of custom **image files** to adapt the appearance of model elements. EMF and MSDKVS differ in their support of various file formats. Icons can be used to, for example, add custom appearances to tool palette items or composition shapes. Different types of **tools** have to be defined to create models in a runtime environment. The granularity of what types of tools can be created and customized varies greatly between EMF and MSDKVS. MSDKVS only allows the definition of essential **element creation tools** for domain classes and domain relationships. On the other hand, EMF offers the definition of a vast amount of additional tools containing, e.g., **edition tools**, **copy-paste tools**, or **reconnect edge tools**. This functionality is not customizable on MSDKVS, but some are automatically used when a creation tool is created. Thus, copying and pasting or deleting elements on the modeling canvas works out of the box.

5 Transformation Bridge

Fig. 3 sketches all three layers involved in realizing interoperability between EMF and MSDKVS. On the left, the MSDKVS column consists of the implicitly defined meta-

metamodel on the M3 layer (see Fig. 2), with its user-defined metamodel on the M2 layer. The metamodels are serialized in XML format as .dsl files. These .dsl files are used as input and output of the transformation, depending on which platform is the source and the target of the transformation. The Transformator itself is divided into transforming metamodels (**M2 Transformator**) and models (**M1 Transformator**). The M2 Transformator is written in C# and de-serializes the incoming files into data structures that can be manipulated and worked with on code level. Abstract and concrete syntax elements represented as XML tags inside these input files are examined, and the mapping rules, based on the M3 concepts of both platforms, are applied sequentially to transform the source metamodel into an equivalent metamodel of the target platform. The greatest challenges faced during the realization of the M2 Transformator are discussed in the following and detailed steps of how these obstacles were solved are given.

5.1 EMF2MSDKVS

Nested EPackage Flattening. We recognized different styles of EPackage definitions in publicly available EMF metamodels (see Table 1 in the Grouping row). Ecore metamodels can either have one or multiple EPackages defined, while EPackages may also have ESubPackages. Therefore, as MSDKVS usually only has one equivalent language definition, these EPackage contents are flattened and merged into one global EPackage before executing the transformation. Naming conventions and avoiding name clashes are transformed accordingly.

Entity Name Clashes. Detecting and resolving name clashes are essential when realizing metamodeling platform interoperability [3]. Different naming strategies to avoid possible name clashes, e.g., across multiple ESubPackages, are executed. For domain relationships, the MSDKVS names are changed as follows:
<sourceEClass.name>_<EReference.name>_<targetEClass.name>.
Name clashes on domain classes (if there are any), are resolved by mapping the EPackages' nsPrefix attribute to the DomainClass' Namespace attribute.

Multiple Inheritance. As EMF, in contrast to MSDKVS, supports multiple inheritance, a transformation of multiple inheritance structures into equivalent single inheritance structures is necessary. We adapted the *Expansion Strategy* pattern proposed by Crespo et al. [10] to translate the complex structures of multiple ESuperTypes in EMF into equivalent single BaseClass references in MSDKVS without information loss (see Fig. 4). Important to note is that also EReferences that target a super class have to be duplicated to the newly created domain classes as Domain Relationships in MSDKVS. In addition to the abstract syntax duplicates, this affects the transformation of all types of graphical concrete syntax mappings from Sirius as well, which results in more Shape classes on MSDKVS side and Creation Tools inside the modeling editor.

Root Element Pattern Matching. MSDKVS metamodels require a root element that is mapped to the diagram shape. This diagram shape provides the modeling canvas in the runtime instances of a domain model. As per API requirement, this selected root element has to be the source domain role of domain relationships marked as containment relationships, where the targets are all domain classes that are neither part of an existing containment relationship (e.g., children of compartments)

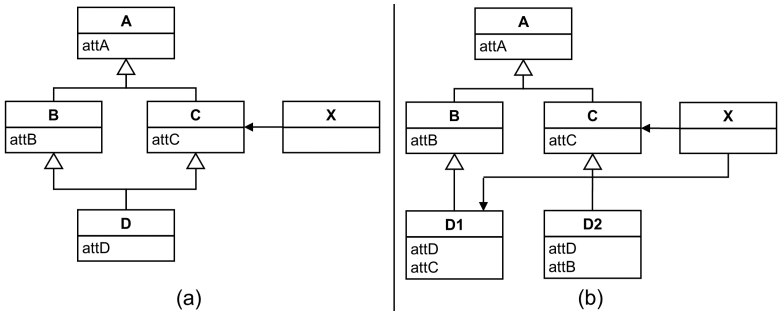


Fig. 4: Adapted *Expansion Strategy* [10]: (a) multiple inheritance in EMF; (b) transformed single inheritance in MSDKVS

nor should target any base classes they would inherit from. When transforming an Ecore metamodel into MSDKVS, existing EClasses are matched against these criteria. If such an EClass can be found, this EClass is transformed to be the root diagram element in MSDKVS. If no EClass is suitable, then an additional default domain class is generated that acts as the diagram's root element.

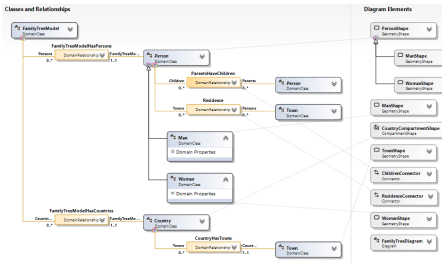
Icon Mapping. Sirius supports the definition of icon styles on different node mappings, referencing workspace images in various image file formats. For MSDKVS, a requirement to attribute a model entity with icons is that the images have to be in the Bitmap (BMP) format. Therefore, library calls to convert these files to the required format on the target platform are employed in the M2 transformation.

5.2 MSDKVS2EMF

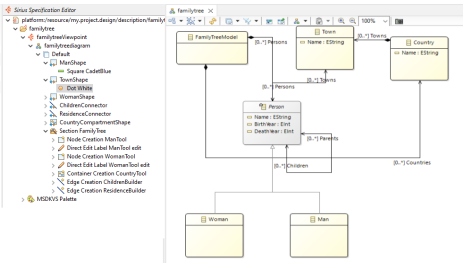
Relationship Roles. Domain Relationships in MSDKVS differ from their required representation on the target EMF side in so far, that the source and the target entities of said relationships are referencing the corresponding domain classes through monikers. Source and target domain roles can have different names attributed to them compared to their actual classes used for creating the domain relationship. This construct has to be considered when transforming from EMF to MSDKVS too, as for every EReference at least one role has to be created in MSDKVS. Domain classes are then referenced through moniker types by their unique names. When transforming from MSDKVS to EMF, the transformator has to look up the source and the target domain classes and transform these DomainClasses into the EReferences' eTypes and eOpposites accordingly.

Attributable Relationships. In MSDKVS not only classes but also relationships can have attributes. As already mentioned in [7], this behavior can be implemented similarly, meaning that domain relationships with attributes attached to them are mapped to classes that are referenced from both transformed domain classes, leading to additional EClass and EReference entities on the target EMF side. Multiplicities are transformed accordingly to maintain the original behavior.

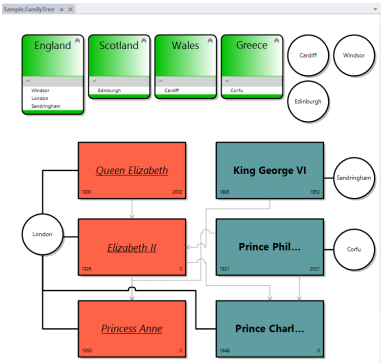
Shape Inheritance. MSDKVS allows inheritance on the graphical representation of classes and relationships. Therefore, the M2 Transformator has to check possible inherited shape classes and transform them accordingly.



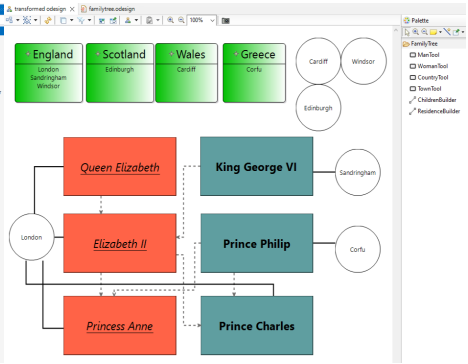
(a) FamilyTree metamodel in MSDKVS



(b) Transformed FamilyTree VSM (left) and metamodel (right) in EMF



(c) A created FamilyTree model in MSDKVS



(d) A created FamilyTree model using the transformed metamodel in EMF

Fig. 5: FamilyTree metamodel and model in MSDKVS and transformed into EMF.

Implicit Modeling Tool Capabilities. MSDKVS supports only the explicit definition of element creation tools on domain classes and domain relationships, while other tooling capabilities that can be explicitly defined in Sirius are inherently available on MSDKVS' modeling canvas. To achieve an equivalent experience, the number of tools on EMF is thus typically higher because the M2 Transformator generates these additional tools for every Node or Edge Creation Tool defined in MSDKVS.

Color Naming. MSDKVS supports a variety of of colors for graphical properties (e.g., FillColor, TextColor, and BackgroundColor). Sirius only supports a small subset of these named colors, e.g., standardized system colors like white, black, and green. To be able to transform said colors from MSDKVS into equivalent colors in EMF, the M2 Transformator looks up the composing red, green, and blue color values for MSDKVS' named colors and transforms them to Custom User Palettes used by Sirius which can be named by the designer.

5.3 MSDKVS2EMF Transformation Example

For showcasing the transformation bridge, we will, in the following, refer to a small example of a family tree metamodel which we created in MSDKVS and subsequently transformed, using our transformation bridge (see Section 5) to a valid EMF metamodel.

The example, adapted from the tutorial of Sirius[4], comes also with a graphical concrete syntax specification on MSDKVS' side, which enables to exemplify feasibility of the transformation bridge. The goal of this example case is thus to show (and illustrate) the feasibility of realizing syntactic and semantic equivalence between the two platforms involving the abstract and the graphical concrete syntax. Full images can be found here[2].

Fig. 5a shows the source metamodel inside the Visual Studio IDE. In this example, a basic family tree metamodel with graphical concrete syntax descriptions has been created that contains a compartment relationship between *Country* and *Town* and an inheritance structure between the *Person* domain class as the base class of *Man* and *Woman*. Fig. 5c shows a manually created model of an excerpt of the family tree of the British House of Windsor based on the previously defined metamodel in MSDKVS.

The result of executing the M2 Transformator on the source MSDKVS metamodel is shown in Fig. 5b, displaying the resulting Ecore metamodel both graphically and in a tree structure. Eventually, Fig. 5d shows a model created manually within a run-time instance that uses the transformed Ecore metamodel and the Sirius .odesign specification. The original model created previously in MSDKVS was used as a reference to show how both appear visually to the modeler, respectively.

6 Evaluation

This section reports on the results of experimenting with the M2 Transformator. The transformation is implemented as two uni-directional transformations which means that either MSDKVS metamodels (*.dsl files) or EMF metamodels (*.ecore files) with optional .genmodel and .odesign files for graphical concrete syntax mappings and code editor generation settings can be used as input.

We searched and selected a representative set of metamodels of both platforms from publicly available collections and also through dedicated metamodel search engines [21]. A collection of 44 metamodels from MSDKVS and 75 randomly selected metamodels from the AtlanMod Atlantic Zoo[5] with additional 22 metamodels referencing and containing Sirius VSMs and 18 metamodels (some of which overlapping with VSM available metamodels) containing EMF specific features like multiple inheritance or nested EPackages was composed. Thus, the metamodels have been selected mostly at random, all manual additions were motivated by the goal to have a representative set of metamodels (i.e., a set that differs in size, contained metamodeling concepts, and are equipped with a graphical concrete syntax specification).

6.1 Research Questions

The aim of the experiments was to evaluate the transformation bridge's overall validity and the success rate, i.e., whether the transformed metamodels are syntactically and semantically equivalent and whether it is possible to transform all source metamodels, respectively. To help with the analysis, additional methods were implemented to create informative output files in addition to the resulting metamodels to list the steps the

[4] `https://wiki.eclipse.org/Sirius/Tutorials/StarterTutorial`
[5] `https://github.com/atlanmod/atlantic-zoo`

Table 1: Source metamodels abstract (left) and transformation success rate (right)

	EMF			MSDKVS		
	Min	Med	Max	Min	Med	Max
Grouping	1	2	20	1	1	1
Classes	1	14	673	2	8	39
Abstract Classes	0	2	83	0	1	9
Inherited Classes	0	7	679	0	3	34
Multiple Inheritances	0	0	134	-	-	-
Relationships	0	12	437	0	9	40
Attributes	0	10	124	1	15	103
Enumerations	0	0	15	0	1	21

Direction	Abstract Syntax			Concrete Syntax		
	Cases	Errors	Rate	Cases	Errors	Rate
MSDKVS → EMF	44	1	97.73%	44	3	93.18%
EMF → MSDKVS	75	0	100%	22	2	90.91%

M2 Transformator has executed and the entities it has created on the target side. After traversing through every metamodel and transforming it, the summarized statistics provide insights on different criteria (see Table 1).

The derived statistics were analyzed and described based on quantitative metrics reported in the literature [2, 11, 17, 20]. Afterwards, a qualitative analysis is concerned with the validation of the transformed metamodels on the target platform by investigating the equivalence of execution functionality (i.e., the modeling support by create, delete, redo/undo tools etc.) of the source and the transformed metamodels in their respective metamodeling platforms.

With the evaluation, we thus aim to respond to the following research questions: *RQ1: Are the transformed metamodels valid when opened in the target platform?*, and *RQ2: Are the transformed metamodels executable, i.e. can editor code be generated and runtime instances successfully started?*

6.2 Experimental Setup

In the following, we dive deeper into the analytical aspects of the metamodel transformation approach. First, quantitative aspects are listed and compared. Table 1 shows statistical information about the experiment's source metamodels' abstract syntax, the statistics of the conrete syntax is provided in the online supplementary material[2]. To analyse the qualitative aspects, the transformed metamodels are opened with the target platform. Both platforms offer automatic validation, meaning that when the project files are opened, their internal structure is validated. Validation errors, if present, are listed accordingly. As the MSDKVS diagram editor on the M2 layer offers the ability to define concrete syntax elements upon the abstract syntax entities, the validation step checks both areas for errors. On EMF side, the transformed .odesign files containing the definitions for graphical concrete syntax mapping required separate, manual validation. If the validation yielded no errors, the platforms' functionalities upon creating models were tested (i.e., starting of runtime instances and creating/editing models).

6.3 Results

RQ1: Validity of transformed metamodels For the transformation validity we not only concern the abstract syntax but also the constraints on the concrete graphical syntax. We encountered only one erroneous case (out of 44) in the direction MSDKVS

→ EMF considering the abstract syntax. This error resulted from the fact that one of the metamodels from MSDKVS did not use the Dsl root tag in its XML serialization. Instead, DslLibrary was observed to be the root element. If changed to Dsl, the transformation executed successfully. The M2 Transformator though has to be extended to also allow DslLibrary as a root XML tag for loading MSDKVS metamodels. Regarding the concrete graphical syntax, two erroneous cases have been identified. One case was caused by the fact that the mapping rule for mapping port shapes to inner mappings in bordered nodes in EMF had not been implemented yet. The second error occurred because of a missing transformation of icon decorator styles for domain properties of compartments.

The direction EMF → MSDKVS yielding a 100% success rate (based on 75 cases) considering the abstract syntax. The additional Sirius validation considered the 22 metamodels which contained a graphical concrete syntax specification. In the runs that followed, two out of 22 were faulty, stemming from the fact that multiple Ecore metamodels were referenced from within the .odesign file, therefore creating shapes for not available entities. In the next steps regarding the M2 Transformator implementation, these errors will be looked at accordingly.

As the M2 Transformator is implemented as a bridge between two XML serialization formats, most of the initial problems that occurred originated from de-serializing the input metamodel files to data structures. XML namespace errors were among the most common types of exceptions because of limitations the default C# library provides when de-serializing Ecore metamodel files, as they contain numerous classes using the same typed attributes (e.g., firstModelOperations and subModelOperations in tool sections). These errors could be eliminated eventually by changing the content of the metamodel files before the de-serialization.

RQ2: Executability of transformed metamodels The executability assumes a valid metamodel in order to run the code generators for creating and executing modeling runtime instances in the platforms. When the code generation throws errors or the modeling canvas cannot be initialized, the transformed metamodel files are deemed faulty with regards to their executability and semantic evaluation. All transformed metamodels, that threw no errors during their validation phase, could be used to generate model code on the target platforms and thus initialize runtime editor instances for modeling purposes. As a final step, a small subset of metamodels on each side and their transformed results were selected for manually evaluating and comparing their interaction on the modeling layer, similarly to how the example in Section 5.3 was conducted. Regarding e.g., their tool palette functionalities, their interaction on the modeling canvas, and their entity styling no major inequalities were observed.

6.4 Limitations

A few limitations adhere to the transformation approach based on the M2 layer. The M2 transformator is specific to the EMF and MSDKVS metamodels. Each additional platform requires a mapping according to the identified rule sets and each rule requires an implementation in C#. For serializing the new metamodel representations, the XML schema has to be analyzed and corresponding data structures for (de-)serialization have to be added. Existing transformations can afterwards be reused, thus creating M3 level

bridges spanning multiple metamodels. Another limitation is the realization of the transformator on the current platform versions. Core updates to these platform metamodels therefore require similar updates of the transformator.

Regarding the involved metamodel entities and unique features of each platform, some limitations in the tranformation's current state are present as well. Many of the advanced model operations supported by Sirius (e.g., filtering using query languages, flow control operations like *Begin*, *For* and *If*, Dialog, and Model Representations operations) cannot be mapped directly to available operations in MSDKVS. As mentioned in Section 4, only element creation tools are supported, which target specific metamodel entities. These features are correctly mapped to and from Sirius' model operations *Change Context*, *Create Instance*, and *Set*.

7 Conclusion

We established interoperability between the EMF and MSDKVS platforms by conceptualizing and implementing a bidirectional transformation bridge able to transform metamodels defined within both platforms. The evaluation was conducted by choosing a representative set of publicly available metamodels[2]. By providing mapping rules also for the graphical notations available in both metamodeling platforms, we showed that it is possible to transform means of graphically representing these elements of transformed metamodels to achieve not only syntactic but also visual interoperability.

Future research concentrates on the realization of the M1 Transformator to transform models between EMF and MSDKVS. Furthermore, the realization of roundtrip transformations based on defined mapping tables and an additional transformation log file is on our agenda. Eventually, we will release the bridge's code and metamodels open source and deploy the transformation bridge as a service at: `http://me.big.tuwien.ac.at/`.

Acknowledgements

This research has been partly funded by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 854187.

References

1. Bézivin, J., Hillairet, G., Jouault, F., Piers, W., Kurtev, I.: Bridging the MS/DSL Tools and the Eclipse Modeling Framework. In: Int. Workshop on Software Factories at OOPSLA (2005)
2. 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)
3. Bork, D., Anagnostou, K., Wimmer, M.: Towards interoperable metamodeling platforms: The case of bridging adoxx and emf. In: Advanced Information Systems Engineering. 34th International Conference, CAiSE 2022. pp. 479–497. Springer (2022)

4. Bork, D., Karagiannis, D., Pittl, B.: Systematic analysis and evaluation of visual conceptual modeling language notations. In: 12th International Conference on Research Challenges in Information Science. pp. 1–11. IEEE (2018). https://doi.org/10.1109/RCIS.2018.8406652
5. Bork, D., Karagiannis, D., Pittl, B.: A survey of modeling language specification techniques. Inf. Syst. **87** (2020). https://doi.org/10.1016/j.is.2019.101425
6. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, Second Edition. Synthesis Lectures on Software Engineering, Morgan & Claypool (2017)
7. Brunelière, H., Cabot, J., Clasen, C., Jouault, F., Bézivin, J.: Towards model driven tool interoperability: Bridging eclipse and microsoft modeling tools. In: 6th European Conference on Modelling Foundations and Applications ECMFA. pp. 32–47. Springer (2010)
8. Bézivin, J., Brunette, C., Chevrel, R., Jouault, F., Kurtev, I.: Bridging the generic modeling environment (gme) and the eclipse modeling framework (01 2005)
9. Cook, S., Jones, G., Kent, S., Wills, A.: Domain-specific development with visual studio dsl tools (05 2007)
10. Crespo, Y., Marqués, J., Rodríguez, J.: On the translation of multiple inheritance hierarchies into single inheritance hierarchies. pp. 30–37 (01 2002)
11. Di Rocco, J., Di Ruscio, D., Iovino, L., Pierantonio, A.: Mining metrics for understanding metamodel characteristics. In: Modeling in Software Engineering. ACM (2014)
12. Geraci, A., Katki, F., McMonegal, L., Meyer, B., Lane, J., Wilson, P., Radatz, J., Yee, M., Porteous, H., Springsteel, F.: IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries. IEEE Press (1991)
13. Hebig, R., Seidl, C., Berger, T., Pedersen, J.K., Wasowski, A.: Model transformation languages under a magnifying glass: a controlled experiment with xtend, atl, and QVT. In: ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. pp. 445–455. ACM (2018)
14. Jouault, F., Bézivin, J.: KM3: A DSL for metamodel specification. In: Gorrieri, R., Wehrheim, H. (eds.) 8th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems. pp. 171–185. Springer (2006)
15. Kern, H.: The interchange of (meta)models between metaedit+ and eclipse emf using m3-level-based bridges. In: 8th Workshop on Domain-Specific Modeling. pp. 14–19 (2008)
16. Kern, H.: Modellaustausch zwischen ARIS und Eclipse EMF durch Verwendung einer M3-Level-basierten Brücke, pp. 123–137 (09 2008)
17. Kern, H., Hummel, A., Kühne, S.: Towards a comparative analysis of meta-metamodels. In: Lopes, C.V. (ed.) SPLASH'11 Workshops. pp. 7–12. ACM (2011)
18. Kern, H., Kühne, S.: Integration of microsoft visio and eclipse modeling framework using m3-level-based bridges. In: Workshop on Model-Driven Tool & Process Integration (2009)
19. Kühne, T.: Matters of (meta-)modeling. Softw. Syst. Model. **5**(4), 369–385 (2006)
20. 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)
21. López, J.A.H., Cuadrado, J.S.: MAR: a structure-based search engine for models. In: 23rd Int. Conf. on Model Driven Engineering Languages and Systems. pp. 57–67. ACM (2020)
22. Microsoft: Official online documentation of the modeling sdk for visual studio (2022), `https://docs.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages`, last accessed on 08.05.2022
23. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: EMF: Eclipse Modeling Framework 2.0. Addison-Wesley Professional, 2nd edn. (2009)
24. Viyović, V., Maksimović, M., Perisić, B.: Sirius: A rapid development of dsm graphical editor. In: Intelligent Engineering Systems INES 2014. pp. 233–238. IEEE (2014)
25. Wimmer, M., Kramler, G.: Bridging grammarware and modelware. In: Bruel, J. (ed.) Satellite Events at the MoDELS 2005 Conference. pp. 159–168. Springer (2005)