Towards Enabling Domain-Specific Modeling Language Exchange between Modeling Tools

Rohit Gupta[1], Christoph Binder[2], Nico Jansen[3], Ambra Calà[1], Jan Vollmar[1], Nikolaus Regnat[1], David Schmalzing[3], and Bernhard Rumpe[3]

[1] Siemens Technology, Munich, Germany
{rg.gupta,ambra.cala,jan.vollmar,nikolaus.regnat}@siemens.com
[2] FH Salzburg, Salzburg, Austria
christoph.binder@fh-salzburg.ac.at
[3] RWTH Aachen University, Aachen, Germany
{jansen,schmalzing,rumpe}@se-rwth.de

Abstract. Domain-specific modeling languages (DSMLs) enable various stakeholders in solving complex modeling problems that are related to their domains. However, as challenges in domain-specific modeling grow in complexity, consistent exchange of domain-specific information between various stakeholders across projects becomes a challenge, as such stakeholders often use a variety of modeling tools suited for their needs. Stakeholders often live within specific modeling tools relevant for developing and using their DSMLs, therefore there is little consideration for generalizing these domain-specific concepts across modeling tools. Further, there also exists a certain lack of exchanging domain-specific constructs effectively for such tools. To solve this challenge, we propose a bi-directional exchange mechanism between Enterprise Architect (EA) and MagicDraw, two commercially established modeling, that allows exchanging individually created DSMLs and their constructs, essential in promoting tool interoperability. As DSMLs represent domain-specific peculiarities within a single area, the proposed exchange mechanism evaluates with a simple illustrative example the applicability of DSML information exchange by extracting and translating these peculiarities across modeling tools. The approach is demonstrated by developing individual extensions to the mentioned tools that support the seamless exchange of domain-specific constructs. Ultimately, the paper presents a first step towards enabling DSML exchange between all the concerned stakeholders and fosters the engineering of DSMLs across modeling tools.

Keywords: Domain-Specific Modeling Language (DSML) · Modeling Tools · Language Exchange.

1 Introduction

Model-driven engineering introduces a system model as a unique source of truth including several viewpoints that address all involved stakeholders and their

[GBJ+24] R. Gupta, C. Binder, N. Jansen, A. Calà,
J. Vollmar, N. Regnat, D. Schmalzing, B. Rumpe:
Towards Enabling Domain-Specific Modeling Language Exchange Between Modeling Tools.
In: Advances in Model and Data Engineering in the Digitalization Era,
M. Mosbah, T. Kechadi, L. Bellatreche, F. Gargouri, C. G. Guegan,
H. Badir, A. Beheshti, M. M. Gammoudi (Eds.), pp. 89–103, ISBN 978-3-031-55729-3,
Springer Nature Switzerland, Cham, Mar. 2024.

cross-cutting concerns. While one model can effectively describe the entire system, it is often broken down into smaller sub-systems that help detail every aspect of the complete system. Stakeholders therefore need to consider the entire product lifecycle to cope with growing complexities in multi-disciplinary modeling projects. To this end, different modeling tools and framework, such as MagicDraw [28], Rational Rhapsody [25], MetaEdit [36], and Enterprise Architect (EA) [16], are used to create various parts of a model within these individual projects [32]. While such tools generally consider and allow the modeling of language constructs, their application in a domain-specific scenario is often challenging in terms of reusability and interoperability. Exchanging artifacts of Domain-Specific Modeling Languages (DSMLs) between proprietary modeling tools is still challenging. While there are studies that detail tool interoperability and model synchronizations between modeling tools [8], applicability of such studies in a real world context is still missing. Consideration for all kinds of stakeholders at all stages of modeling using a DSML is often lost in translation, as DSML projects are often time-bound and only exist in a single modeling environment. Little consideration is made for developing and reusing metamodels with the idea of a language isomorphism across different modeling environments in mind (section 2). In this paper, we share our experiences in establishing a bidirectional exchange mechanism between two established commercial modeling frameworks, MagicDraw and EA, in an industrial context. These tools have been chosen as they are established within the systems engineering community and the need for exchanging artifacts has appeared in actual industrial projects [8]. Further, these tools are primarily based on Unified Modeling Language (UML) standards and can be extended to include domain-specific aspects for a variety of domains. The introduced exchange mechanism enables the synchronization of DSMLs across modeling environments by exchanging the respective modeling language artifacts, where the language definition in an individual tool is extracted and translated to a format reusable by the other tool. An advantage of this technique is that stakeholders can benefit from such an interchange of the DSML across modeling tools, projects, and organizations. The use of such a mechanism allows language engineers to effectively exchange similar concepts of a DSML across multiple tool and do not need to develop DSMLs always from scratch. However, in contrast to other promising model transformation approaches like ATLAS Transformation Language (ATL) [26], our approach does not mainly target exchanging of finished models but rather transforming its meta-information, such as the UML Profile, which can be used for jointly creating those models across a variety of tools. We realize the implementation of the mechanism using custom plugins written in Java that enhance and extend the functionalities of the respective tools. While the proposed approach targets the mentioned modeling frameworks and represents a proof-of-concept (PoC) for introducing such a mechanism, it is generally applicable to similar modeling tools that support language development and where the exchange of a DSML is required. The concepts described in this study takes a first step in ensuring tool interoperability and consistent DSML development views between language en-

gineers across multi-disciplinary modeling projects and modeling environments. The main contributions of the paper, given our expertise in language engineering in the industry and the academia as well as with our experience in developing DSMLs with multiple modeling tools and language workbenches, are as follows:

- We detail a concept for a bidirectional exchange mechanism (section 4) for enabling tool interoperability for the exchange of DSML constructs using the modeling tools MagicDraw and EA (section 5).
- We discuss the extension mechanisms of the modeling tools MagicDraw and EA (section 3) that form the underlying basis of the implementation of the exchange mechanism, along with related approaches (section 8).
- We discuss using a simple example of use cases, actors, and their tasks (section 6), how the concepts of the described exchange mechanisms can be generally transferred to other modeling tools (section 7), and finally conclude the paper in section 9.

2 Background

To easily manage and evolve complex systems across business units and organizations, it is important to build modular reusable DSML elements that can be easily interchanged between modeling tools. Often large organizations have multiple business units that use different modeling tools to realize and solve similar challenges in their domains. On the other hand, small and medium enterprises are limited to using a single modeling tool in a single modeling environment, requiring extensive engineering efforts in developing conceptually similar DSMLs. While efforts have been made to enable the globalization of Domain-Specific Languages (DSLs) using model-based engineering approaches [10], the engineering of DSMLs is still challenging in organizations that require support for development and usage of DSMLs in multiple modeling environments. The exchange of models and language constructs between various model-driven software and systems development tools and workbenches require a large effort [34]. Generally, language engineers and modelers operate within the ecosystem of their modeling environments and do not specifically consider the aspect of reusability of a DSML and its constructs beyond the boundaries of the respective technological spaces. However, more recently the Object Management Group (OMG) has come up with ways to address challenges of exchanging models in a more standard way, such as by using XML Metadata Interchange (XMI) file formats [3]. While this is beneficial in environments that use a more vanilla form of UML, considerations for exchanging models between DSML environments is still lacking. Modeling tools do not particularly generate a standard form of exporting and importing domain-specific constructs for similar models, often leading to issues determining the model elements' semantics in such exchange formats. As an example, if we model UML stereotypes with the tool MagicDraw and export it to an XMI format, it generates elements configured with a metaclass *uml:Stereotype*, while the same model generates a plain UML metaclass *uml:Class* when exported from EA. Such inconsistencies in the underlying stored format of a DSML hinders the

reuse of constructs between modeling environments and delays the successful deployment of DSMLs across organizations. It is important to align the metamodels in different modeling tools such that the mapping of DSML constructs is beneficial in easily interchanging and reusing domain concepts across modeling environments.

3 DSML Specification in Modeling Tools

3.1 MagicDraw

MagicDraw is primarily based on UML and supports customization possibilities to enhance the modeling experience of DSML users [21] through modular reusable building blocks of DSMLs [22]. These building blocks consist of reusable language components, that define the language wholly or in part [34]. The customization capabilities of MagicDraw can be leveraged to create a language profile consisting of language component artifacts. An example of such an artifact is a language element, referred to as a *stereotype* in MagicDraw. For each *stereotype*, a set of *customizations* can be configured that are, in essence, rules to define where model elements can be created and which specific properties of the model elements can be displayed to users. MagicDraw supports Java Open API that allows language engineers to integrate automation and creation of custom functionalities that are not supported by default. Model templates allows for the creation of a predefined model that is automatically instantiated during design time. Perspectives in MagicDraw help display various tool and DSML functionalities that users require based on their skill level, novice or advanced. The language profile of the DSML is exported as an Extensible Markup Language (XML) file which allows for a language definition to be reused across other projects and be translated to other modeling tools. The final archived plugin file consists of an additional descriptor XML file that references all the language artifacts and loads the source files into memory so that the DSML is ready available to modelers on tool start-up. In various modeling projects such as in [7], we naturally encountered challenges in translating language profiles [35] across modeling tools with heterogeneous domain concepts as well as in bridging the gap between textual and graphical modeling.

3.2 Enterprise Architect

Within EA, a DSML is implemented in the form of a model-driven generation (MDG) technology [30] file. This term is the EA-specific representation of an individual modeling notation. Thereby, the core concept of the DSML is the UML metamodel, which represents the conceptual architecture and the interconnection of the domain-specific elements. By doing so, the metamodel inherits all elements needed to describe the intended application domain and shows the dependencies with the real world. The MDG allows the extension of EA's modeling capabilities, similar to the customizations in MagicDraw, and consists of three main parts: (1)

UML Profile: the UML Profile consists of all domain-specific elements including their UML metaclass and relationships or restrictions; (2) *Diagram Profile*: the Diagram Profile selects the corresponding UML diagram on which the DSML is built and maps a toolbox to this diagram. Each diagram only allows modeling with a particular set of elements; and (3) *Toolbox Profile*: within the Toolbox Profile, all UML elements are classified within groups according to a modeling context. A natural distinction is separating elements and relationships. Apart from these parts, additional information could be stored in the MDG-file. This information could consist of alternative representations of model elements by selecting suitable images or additional attributes for those elements. As those are not part of the UML Profile, they are additionally exported and stored in an XML file. Ultimately, the file is stored within the MDG Technologies folder of the EA instance.Within this folder, any other DSMLs, such as those implementing the Zachman framework [37], that have been developed and introduced, are stored for provisioning. However, when storing the created MDG into this folder, its information is loaded directly with the start-up of EA and hence no additional setup needs to be performed. Diagrams including the mapped model elements are directly added via the context menu and used for modeling domain-specific system architectures.

4 Concept for a DSML Exchange Mechanism

The capabilities of both tools set the precedent for deriving a generic concept for the transformation of DSMLs between modeling tools as described in Figure 1. The general concept of exchanging DSML constructs is through: (1) a modeling tool extension that imports and exports DSML constructs and their information through; (2) a standard file format and; (3) by identifying the appropriate meta-classes. Here, the concept details both the export and import processes from one modeling tool to another.

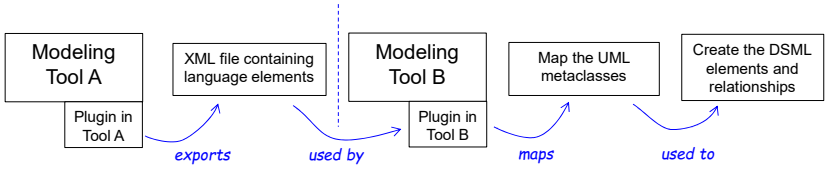

Fig. 1. A general concept for interchanging DSML elements between modeling tools.

Certain customizations are possible in modeling tools that extend their functionalities. This can be achieved through adding a custom code in any General Purpose Language (GPL) in the form of tool plugins or add-ins. Here, modeling tool *A*'s plugin extracts language elements and their properties, such as symbols and relations into an XML formatted file. This is done through Application

Programming Interfaces (APIs) exposed by the modeling tools. Modeling tool B's plugin then imports the XML file and provides capabilities to parse, map, and create the DSML elements. The exported XML file is parsed by B's plugin. This XML file consists of both UML and DSML constructs that convey the necessary information about the various elements of a DSML. In this way, semantic properties of a construct can also be defined as part of the exchange. Next, B's plugin parses the XML file from A and maps the individual elements to its appropriate UML metaclass. This mapping ensures that metadata information is not lost during the exchange process. Finally, the mapped data is translated into DSML elements through a UML profile. B's plugin creates elements and their properties on the profile through APIs so that the DSML consists of the language profiles, elements, and their relationships exported from A to B.

5 Implementation

5.1 Plugin for MagicDraw

While XMI allows defining a common format for standard UML models, it has limitations in encoding accompaniment information for domain-specific constructs, making the interchange of DSML information in modeling tools challenging. Domain-specific XMI models exported from EA contain XML attributes and EA-specific data that is incomprehensible to default MagicDraw import mechanisms. As discussed in section 3, custom functionalities can be added in the form of additional plugins bundled within the final DSML profile in MagicDraw. To this end, we develop a plugin, using MagicDraw's API, to enable importing a language profile and its constructs from EA. The plugin adds an additional context menu item to MagicDraw's default import mechanism for selecting an EA exported XMI file to extract information from it. Once this information is extracted and translated, the respective stereotypes and their customizations are configured and created on a MagicDraw profile. The *referentPath* is the attribute exported from MagicDraw and used for import into EA.

The plugin in MagicDraw imports the EA-XMI file that is constructed in a way that the elements of the language are configured under the XML nodes *uml:Model* and *packagedElement*. An example of a EA-XMI file is shown in Listing 1.1, where the identifiers (*xmi:id*) for each node is modified to illustrate the relations to the respective language elements. The elements from the XMI nodes are stored in custom objects. Next, MagicDraw's OpenAPI is used to create a DSML project, a language profile, and a UML profile diagram, for setting the stereotypes, their customizations and metaclasses. To correctly configure the stereotypes, the extracted information must be translated for the domain-specific classifiers. As there is a possibility for XMI conflicts such as with domain-specific classifiers, special cases must be handled by the plugin, such as identifying the correct metaclasses for a specific *xmi:type* attribute. For example, a *uml:Task* type is translated to a *Class* metaclass, as in a domain-specific context we consider tasks as activities in a UML activity diagram to be the most general UML metaclass. On the other hand, a *uml:Association* type is directly extracted and

Listing 1.1. An example of an (incomplete) XML file generated in EA detailing a DSML with use cases, actors, tasks, and their relations.

```
<uml:Model xmi:type="uml:Model" name="EA_Model">
<packagedElement xmi:type="uml:Package" xmi:id="pkg_ID" name="Package">
<packagedElement xmi:type="uml:Actor" xmi:id="actor_ID" name="Actor"/>
<packagedElement xmi:type="uml:UseCase" xmi:id="uc_ID" name="UseCase"/>
<packagedElement xmi:type="uml:Association" xmi:id="assoc_ID" name="Performs">
<memberEnd xmi:idref="dst_ID"/>
<ownedEnd xmi:type="uml:Property" xmi:id="dst_ID" association="assoc_ID"/>
<memberEnd xmi:idref="src_ID"/>
<ownedEnd xmi:type="uml:Property" xmi:id="src_ID" association="assoc_ID"/>
</packagedElement>
<packagedElement xmi:type="uml:Task" xmi:id="task_ID" name="Task"/>
</packagedElement>
</uml:Model>
```

used, since MagicDraw also uses the same classifier for an association connector. For an association, there must also be restrictions set on the configured stereotypes that allows associations to be created only between certain stereotypes. In this case, the *ownedEnd* nodes of a *packagedElement* in the XMI is first extracted and then the identifiers that specify the source and destination ends of the association are looked up. These identifiers are used to set the *typesForSource* and *typesForTarget* customization properties in MagicDraw that help restrict the use of a connector between specific model elements. A limitation of this import process is that icons for the model elements cannot be stored directly within the XMI file, but have to be transmitted and processed individually outside of the XMI extraction process. Then the stereotypes are created within the language profile. Once all the artifacts are generated, they are stored in a *.mdzip* file, an archive file storing all the information related to the DSML, including any additional plugins. This also includes icons, as scalable vector graphic images, and any model templates that can be provided to modelers for quickly creating models. This DSML profile is now installed in the modelers' MagicDraw environment. Language engineers can, therefore, configure a family of similar DSML profiles across both MagicDraw and EA. MagicDraw provides functionality to export the entire language profile as XMI, which we now use to showcase the import process from MagicDraw to EA.

5.2 Add-In for Enterprise Architect

To enable export and import of DSML information, an Add-In for EA, the so-called RAMI Toolbox [6], is expanded in functionality. More specifically, the RAMI Toolbox allows the engineering of complex production systems for various industrial domains. The toolbox is used for exporting and importing such elements making it ideal to be applied in the context of this paper. Exporting domain-specific information from EA is straightforward, as the MDG file is exported in XML format. This is done as the XML stores the needed information within a single file that requires extraction. However, importing domain-specific elements into EA via the RAMI Toolbox is more complex, as information needs to be extracted from the MagicDraw XMI file that contains the modeled elements. Those elements must then be properly integrated within an MDG file

that can be interpreted by EA. We extract the information out of the MagicDraw XMI file to find all domain-specific elements. These are embedded in the *packagedElement* XML nodes. The RAMI Toolbox deals with iterating through the file and storing all such elements for further processing. In general, we consider the elements classified as *uml:Stereotype* to represent a domain-specific element. Within such a stereotype, the name could be easily extracted via its same-called attribute. However, the underlying UML class is of major importance, as this attribute defines the underlying foundation of each element. This means, the third attribute deals as a standardized exchange format between both frameworks, MagicDraw as well as EA, since DSMLs in either of them is based on UML standard types. Apart from those attributes, relationships between the elements also need to be exported from the MagicDraw XMI. To do so, the attribute *DSL_ Customization* is parsed within the file. This node also contains three attributes, the type of relationship, the source element, and the target element. Within the relationship type, the *customizationTarget* indicates the relationship used between the DSML elements. Next, the *typesForSource* and *typesForTarget* specify the source and target element. When parsing this information, only the specified relationship is used to interconnect the DSML elements within the model. Finally, after exporting this information, it needs to be embedded within an EA-MDG file. More specifically, the parsed DSML elements including their UML related aspects are stored within the UML Profile of the MDG, where the Diagram Profile is adjusted and the toolbox is finally linked for utilization.

6 Application

We illustrate the applicability of our research with an illustrative example involving actors, tasks, and use cases. The goal is to model individual use cases, actors, tasks, and their relationships independently in both EA and MagicDraw using the bi-directional exchange mechanism described in section 5. A use case language is used to model the high-level functionality of a system. An actor language can be used to model actors, their tasks, and the relations between their tasks. In this example, we combine both the use case language and the actor language into a single language, to demonstrate the how the DSML constructs for this language can be translated from EA to MagicDraw. The MagicDraw plugin allows users to import an exported XMI file from EA. This XMI file consists of a language definition with use cases similar to UML use cases, actuators, and their tasks and an example of this XMI file was listed in Listing 1.1. Generally, use cases, actors, and associations are directly translated into their respective UML metaclasses. The plugin extracts the domain-specific constructs from the EA XMI file and creates the respective stereotypes and their customizations in MagicDraw on a UML profile diagram. Figure 2 shows the constituents of this profile diagram consisting of the various stereotypes for the use case, task, actor, association, and their configured properties that have been imported from EA.

While the use case, actor, and association elements have been assigned their respective stereotypes, the domain-specific task element from EA is assigned the

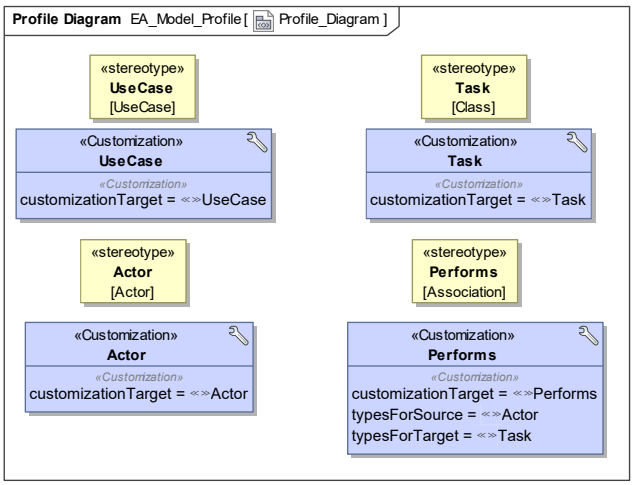

Fig. 2. A MagicDraw UML Profile Diagram for the extracted EA DSML elements.

class metaclass. With a standard XMI file, either the domain-specific constructs or some of their properties cannot be directly interchanged from EA to MagicDraw. To solve this challenge, the plugin is configured to identify constructs that are domain-specific and performs the necessary translation for import into MagicDraw. While the constraints are mainly embedded into properties within MagicDraw elements, EA uses so-called metarelationships or metaconstraints to restrict the misuse of the modeling elements. Thus, the performs relationship directly specifies source and target elements, which is stored into the metaconstraint. To only allow this connector to be used for the actor and task elements, this information is placed within the metarelationship. In order to ensure interoperability between EA and MagicDraw, the plugin deals with translating each other's information and transforms the properties into connectors and vice versa. The plugin is also responsible for calculating the references between the elements based on the XMI file and configures those references, such as setting the source and target elements for a relationship. In doing so, EA can also generate further information specific to an element, and that would be parsed and utilized by the MagicDraw plugin. Similarly, the XMI file exported from MagicDraw, containing the DSML definition, can be imported into EA. On successful import, the DSML elements are embedded into the EA-MDG file and the required information is stored within the UML profile in EA.

6.1 Model Example

Figure 3 shows an example model created in MagicDraw using the DSML constructs that are imported from EA. Here, the model elements are created individually and the respective stereotypes are assigned to each of these elements. This can be done automatically with a model template, that is created for the imported UML profile and configured with the final archived MagicDraw DSML

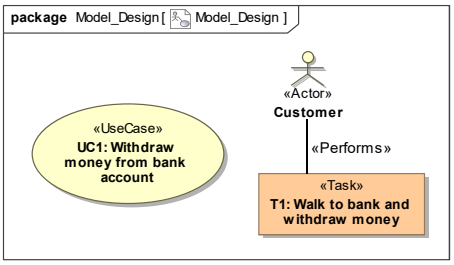

Fig. 3. An exemplary MagicDraw model using the DSML elements configured in Figure 2 showing an actor performing a task for a respective use case.

plugin. Therefore, the *T1* model element, configured as a UML *Class* metaclass, is assigned the *Task* stereotype that is imported from the EA XMI file. Similarly, the *Customer* model element is configured as a UML *Actor* metaclass and receives the stereotype *Actor*. An association between *Customer* and *T1* is assigned the *Performs* stereotype, which is a UML *Association* with the configured source and target stereotypes. Finally, the *UC1* model element is configured with the UML *UseCase* stereotype. The illustrated model can also be replicated in EA using the same DSML exported from MagicDraw as an XML file.

7 Discussion

The approach presented in this paper to solve the challenges of DSML interoperability between modeling tools was conducted with a group of practitioners and researchers. The challenge to enable the exchange of DSML and its constructs was identified as a key research topic in improving the overall DSML engineering process across different research units. While exchanging standard UML constructs across different modeling environments is possible via XMI file formats, achieving the same in a domain-specific environment is currently lacking. Therefore, the presented methodology brings a first perspective using two popular commercial modeling tools, EA and MagicDraw. Previously, language engineers had to build DSMLs that represent similar domain concepts from scratch as concepts that detail the reuse DSML of constructs were lacking in the literature. Although the OMG tried to solve this challenge by creating a common file interchange mechanism, with XMI, it did not consider domain-specific aspects which are crucial to effectively realize Model-based Systems Engineering (MBSE) methodologies. Further, different modeling tools often generate such XMI file formats that are unusable in another modeling environment. Extending the existing modeling tools creates the opportunities for language engineers to design DSMLs in a way that their constructs are now reusable. We developed two Java plugin extensions in each of the modeling tools to showcase the bidirectional interchange of DSML constructs using an example of use cases, actors, tasks, and their relations. The solution enables both standard and domain-specific constructs to be translated across business lines that work on different tools

and fosters the reusability of DSML constructs without losing domain-specific information. The presented approach in this paper separates the concerns of collecting DSML constructs in an XML file, parsing those XML files using GPL code, and finally using the tool capabilities to create the language. This separation allows language engineers to easily adopt this solution to their modeling environments as it considers standard file formats that are frequently used [4]. Although our study is conducted in a vendor-locked scenario which introduces a threat to the study, the implementation is described in a way it is seamlessly transferable to other modeling tools that support language development. The use of an XML format and Java code to extend a modeling tool's functionalities makes the solution independent of a modeling tool. This is based on the assumption that a modeling tool exposes APIs, e.g., Java OpenAPI in our case, that allows DSML information such as language elements, models, and their properties to be accessed using an independent GPL code. Although maintaining the plugins require additional effort, we deem this effort low, as modifications to the plugins depend primarily on updates to the API specifications of the involved modeling tools. The applicability of the research using a simple illustrative example demonstrates the capability of the solution to exchange both UML and domain-specific constructs, and can be extended to other modeling tools, or even to other modeling languages such as with SysML. We are currently working to further validate our research in other modeling environments, such as Rational Rhapsody. Therefore, we believe the concepts presented in this paper will encourage language engineers to develop extensions that enable an easy interchange of a DSML and its constructs between various modeling environments that eventually reduces the effort needed in modeling a family or variants of DSMLs across complex modeling environments.

8 Related Work

One technique of interchanging constructs of a modeling language is using standard file formats such as XMI or XML [1,2]. However, with every MBSE approach [10], it is often very challenging to engineer [11], exchange [24], and reuse standard file formats to model domain-specific elements in a variety of domains and across various modeling tools or language workbenches [23,15]. There still exist several challenges on model interchange between model-driven software and systems development tools [34], as the methods and concepts are often only described for a specific purpose and do not consider aspects for generalization. Modeling tools often use different file formats and data structures to store their language concepts such as UML or the Systems Modeling Language (SysML) [13]. The OMG has introduced XMI file formats [3] to exchange models in a more standard way and foster model-to-model transformations such as by using query/view/transformation (QVT) techniques. However, not all tools generate a standard format for exporting domain-specific elements for similar models [38], meaning semantics of the exchange formats are not clearly specified [19,29]. This means that the MBSE tools often lack of conformance to OMG

standards and are strictly designed to consider only plain UML concepts. Existing work on bridging the gap between different technological spaces focuses on translating languages between different technological spaces [9,12], improving modeling language variability [20], and the interoperability of models and language elements [14,5,15]. Additionally, techniques for extending DSLs, which are embedded in a predefined tool, via tagging languages [18], tool integration frameworks [31], or through model federation [17] exist. However, realizing a seamless exchange of DSML constructs between different model-driven development tools in a real-world context is still challenging as it requires domain-specific knowledge across multiple modeling environments that involves a large number of stakeholders. Even though tools such as EA and MagicDraw are powerful for exchanging UML constructs [33,27], truly achieving interoperability, exchange of models, and DSML constructs across different modeling tools needs special consideration. This paper therefore describes how we aimed to reduce this gap and show how we enable DSML exchange between two commercial modeling tools, with the basis that the implementation can be extended to other modeling tools.

9 Conclusion

As systems grow more complex and heterogeneous, so does challenges in the interchange of domain-specific constructs between modeling tools that support DSML development. Standard file interchange formats such as XMI have emerged to solve challenges to exchange metadata of UML models. However, different modeling tools generate different XMI file formats for the concerned domain-specific constructs preventing a seamless language exchange between modeling environments. To solve this challenge, we build an exchange mechanism between two commercial modeling tools, Enterprise Architect and MagicDraw, that allows standard UML and DSML constructs to be interchanged. We create Java-based plugins in the modeling tools that extracts domain-specific information and creates language definitions in the respective tools. This allows language and model elements to be exchanged across different modeling environments without losing domain-specific information. Cross-functional teams across different organizations can therefore reuse language definitions for a single domain. Although this paper constitutes a first step towards DSML exchange between modeling tools, further work is ongoing to validate the exchange mechanism in other modeling tools. Ultimately, such a DSML exchange mechanism allows for the interoperability of DSML constructs in complex modeling environments.

References

1. Extensible Markup Language (XML) (2023), `https://www.w3.org/XML/`
2. Model Interchange Wiki (2023), `https://www.omgwiki.org/model-interchange/doku.php`
3. XML Metadata Interchange (XMI) (2023), `https://www.omg.org/spec/XMI/`

4. Bézivin, J.: Model driven engineering: An emerging technical space. Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. pp. 36–64 (2006)
5. Bézivin, J., Brunelière, H., Cabot, J., Doux, G., Jouault, F., Sottet, J.S.: Model driven tool interoperability in practice. In: 3rd Workshop on Model-Driven Tool & Process Integration (co-located with ECMFA 2010). pp. 62–72 (2010)
6. Binder, C., Neureiter, C., Lüder, A.: Towards a domain-specific approach enabling tool-supported model-based systems engineering of complex industrial internet-of-things applications. Systems **9**(2) (2021)
7. Böhm, W., Broy, M., Klein, C., Pohl, K., Rumpe, B., Schröck, S. (eds.): Model-Based Engineering of Collaborative Embedded Systems. Springer (January 2021)
8. Bruneliere, 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, Paris, France, June 15-18, 2010. Proceedings 6. pp. 32–47. Springer (2010)
9. Butting, A., Jansen, N., Rumpe, B., Wortmann, A.: Translating Grammars to Accurate Metamodels. In: International Conference on Software Language Engineering (SLE'18). pp. 174–186. ACM (2018)
10. Cheng, B.H.C., Combemale, B., France, R.B., Jézéquel, J.M., Rumpe, B.: On the Globalization of Domain Specific Languages. In: Globalizing Domain-Specific Languages. pp. 1–6. LNCS 9400, Springer (2015)
11. Clark, T., Brand, M.v.d., Combemale, B., Rumpe, B.: Conceptual Model of the Globalization for Domain-Specific Languages. In: Globalizing Domain-Specific Languages. pp. 7–20. LNCS 9400, Springer (2015)
12. Dalibor, M., Jansen, N., Kästle, J., Rumpe, B., Schmalzing, D., Wachtmeister, L., Wortmann, A.: Mind the Gap: Lessons Learned from Translating Grammars Between MontiCore and Xtext. In: International Workshop on Domain-Specific Modeling (DSM'19). pp. 40–49. ACM (October 2019)
13. Dalibor, M., Jansen, N., Michael, J., Rumpe, B., Wortmann, A.: Towards Sustainable Systems Engineering-Integrating Tools via Component and Connector Architectures. In: Jacobs, G., Marheineke, J. (eds.) Antriebstechnisches Kolloquium 2019: Tagungsband zur Konferenz. pp. 121–133. Books on Demand (February 2019)
14. Diallo, P.I., Champeau, J., Lagadec, L.: A model-driven approach to enhance tool interoperability using the theory of models of computation. In: Software Language Engineering: 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26-28, 2013. Proceedings 6. pp. 218–237. Springer (2013)
15. Drux, F., Jansen, N., Rumpe, B., Schmalzing, D.: Embedding Textual Languages in MagicDraw. In: Modellierung 2022 Satellite Events. pp. 32–43. Gesellschaft für Informatik e.V. (June 2022)
16. Enterprise Architect (2023), `https://sparxsystems.com/`
17. Golra, F.R., Beugnard, A., Dagnat, F., Guerin, S., Guychard, C.: Using free modeling as an agile method for developing domain specific modeling languages. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems. pp. 24–34 (2016)
18. Greifenberg, T., Look, M., Roidl, S., Rumpe, B.: Engineering Tagging Languages for DSLs. In: Conference on Model Driven Engineering Languages and Systems (MODELS'15). pp. 34–43. ACM/IEEE (2015)
19. Grönniger, H., Ringert, J.O., Rumpe, B.: System Model-based Definition of Modeling Language Semantics. In: Proc. of FMOODS/FORTE 2009, LNCS 5522. Lisbon, Portugal (2009)

20. Grönniger, H., Rumpe, B.: Modeling Language Variability. In: Workshop on Modeling, Development and Verification of Adaptive Systems. pp. 17–32. LNCS 6662, Springer (2011)
21. Gupta, R., Jansen, N., Regnat, N., Rumpe, B.: Design Guidelines for Improving User Experience in Industrial Domain-Specific Modelling Languages. In: Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings. ACM (October 2022)
22. Gupta, R., Kranz, S., Regnat, N., Rumpe, B., Wortmann, A.: Towards a Systematic Engineering of Industrial Domain-Specific Languages. In: 2021 IEEE/ACM 8th International Workshop on Software Engineering Research and Industrial Practice (SE&IP). pp. 49–56. IEEE (May 2021)
23. Hölldobler, K., Rumpe, B.: MontiCore 5 Language Workbench Edition 2017. Aachener Informatik-Berichte, Software Engineering, Band 32, Shaker Verlag (2017)
24. Hölldobler, K., Rumpe, B., Wortmann, A.: Software Language Engineering in the Large: Towards Composing and Deriving Languages. Computer Languages, Systems & Structures **54**, 386–405 (2018)
25. IBM Rhapsody (2023), `https://www.ibm.com/products/systems-design-rhapsody/`
26. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I., Valduriez, P.: Atl: a qvt-like transformation language. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. pp. 719–720 (2006)
27. Kern, H.: Study of interoperability between meta-modeling tools. In: 2014 Federated Conference on Computer Science and Information Systems. pp. 1629–1637. IEEE (2014)
28. MagicDraw Enterprise (2023), `https://www.3ds.com/products-services/catia/products/no-magic/magicdraw/`
29. Maoz, S., Ringert, J.O., Rumpe, B.: Semantically Configurable Consistency Analysis for Class and Object Diagrams. In: Conference on Model Driven Engineering Languages and Systems (MODELS'11). pp. 153–167. Springer (2011)
30. MDG Technologies (2023), `https://sparxsystems.com/resources/mdg_tech/`
31. Mustafiz, S., Denil, J., Lúcio, L., Vangheluwe, H.: The FTG+PM Framework for Multi-Paradigm Modelling: An Automotive Case Study. In: Proceedings of the 6th International Workshop on Multi-paradigm Modeling. pp. 13–18 (2012)
32. Odukoya, K.A., Whitfield, R.I., Hay, L., Harrison, N., Robb, M.: An architectural description for the application of mbse in complex systems. In: 2021 IEEE International Symposium on Systems Engineering (ISSE). pp. 1–8. IEEE (2021)
33. Ozkaya, M.: Are the uml modelling tools powerful enough for practitioners? a literature review. IET Software **13**(5), 338–354 (2019)
34. Rumpe, B.: Modeling with UML: Language, Concepts, Methods. Springer International (July 2016)
35. Gupta, R., Jansen, N., Regnat, N., Rumpe, B.: Implementation of the SpesML Workbench in MagicDraw. In: Modellierung 2022 Satellite Events. pp. 61–76. Gesellschaft für Informatik (June 2022)
36. Tolvanen, J.P.: Metaedit+ integrated modeling and metamodeling environment for domain-specific languages. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications (2006)
37. Zachman, J.A.: The zachman framework for enterprise architecture. Primer for Enterprise Engineering and Manufacturing.[si]: Zachman International (2003)
38. Zusane, U.I., Nikiforova, O., Gusarovs, K.: Several issues on the model interchange between model-driven software development tools (2015)