Towards Model Driven Tool Interoperability: Bridging Eclipse and Microsoft Modeling Tools

Hugo Bruneliere, Jordi Cabot, Caue Clasen, Frédéric Jouault, Jean Bézivin

▶ To cite this version:

Towards Model Driven Tool Interoperability:
Bridging Eclipse and Microsoft Modeling Tools

Hugo Brunelière, Jordi Cabot, Cauê Clasen, Frédéric Jouault, and Jean Bézivin

AtlanMod (INRIA - École des Mines de Nantes) – France
{hugo.bruneliere, jordi.cabot, caue.avila_clasen, frederic.jouault,
jean.bezivin}@inria.fr

Abstract. Successful application of model-driven engineering approaches requires interchanging a lot of relevant data among the tool ecosystem employed by an engineering team (e.g., requirements elicitation tools, several kinds of modeling tools, reverse engineering tools, development platforms and so on). Unfortunately, this is not a trivial task. Poor tool interoperability makes data interchange a challenge even among tools with a similar scope. This paper presents a model-based solution to overcome such interoperability issues. With our approach, the internal schema/s (i.e., metamodel/s) of each tool are explicited and used as basis for solving syntactic and semantic differences between the tools. Once the corresponding metamodels are aligned, model-to-model transformations are (semi)automatically derived and executed to perform the actual data interchange. We illustrate our approach by bridging the Eclipse and Microsoft (DSL Tools and SQL Server Modeling) modeling tools.

1 Introduction

Development of a software system involves the collaboration of many developers with different roles (managers, analysts, designers, programmers,...) employing various tools (from project management tools, as Microsoft Project, to tools for requirements elicitation, as DOORS or even Excel, modeling tools as EMF and Microsoft DSL Tools, and development IDEs among many others).

Clearly, a key aspect for this collaboration is proper interoperability in the tool ecosystem. Interoperability is the ability of two (or several) tools to exchange information and thus to use the exchanged information [11][22]. Interoperability is required in several scenarios: forward engineering, reverse and round-trip engineering, tool and language evolution (to address backward compatibility with previous versions) and, for instance, collaborative development, where several subteams may work on separate views of the system using different tools (e.g., modeling tools) that must be later merged.

Unfortunately, interoperability is also a challenging problem that requires addressing both syntactic and semantic issues since each tool may use a different syntactic format to store its information but, more importantly, use its own internal schema to represent and manipulate such information, most likely different from the one expected by other tools. Therefore, trying a manual solution is

error-prone and very time-consuming, and it is hardly reusable even when using a similar set of tools. Instead of ad-hoc solutions, a generic set of bridges between the tools should be provided. Each bridge should ensure data-level interoperability (i.e., metadata/data interchange) and operational-level interoperability (i.e., behavior interchange) for two or more tools, independently of the specific project/context in which the tools are used.

In this sense, we propose a model-driven solution for tool interoperability. In general, model-driven interoperability approaches work by first making explicit the internal schema (i.e., metamodel) of each tool. Metamodels are then aligned by matching the related concepts. Finally, model-to-model transformations exploit this matching information to export data (i.e., models from our point of view) created with the first tool to data conforming to the second tool's internal schema. In this paper, we focus on a more general scenario in which tools that need to interoperate are able to manipulate data conforming to different metadata specifications. In this situation, data interoperability needs to interchange not only the data but also the metadata between the tools so that the target tool can correctly interpret the imported information. Therefore, in this case, alignment is not done at the metamodel level but at the metametamodel level. Note that tools that support arbitrary metadata specification necessarily represent metadata using a specific format, or structure, and cannot have fully hard-coded metadata. This format is what we refer to as the metametamodel of the tool, whether it is called a metamodel in the tool terminology or not, and whether it is explicit in the tool or not. Also note that, in contrast to other approaches, changes on the internal schema/s used by one of the tools do not require updating the bridge.

We believe this more generic approach is required to deal with the complexity of current model-driven engineering (MDE) approaches. As an example, consider the Eclipse Modeling Framework (EMF [3]). When modeling a system with EMF, designers can use several domain-specific languages, each one represented by its corresponding metamodel, to specify different views of the system. When exporting this specification to another tool we need to export both the models and the metamodels the designers have used. We will use this scenario to illustrate our interoperability approach. In particular, we will provide a set of bridges between the Eclipse (EMF) and Microsoft (SQL Server modeling [5] and DSL Tools [4]) modeling technologies. The bridges will allow to automatically open and manipulate in Microsoft tools any model and metamodel defined in EMF and vice-versa.

As we will see, our model-based solution offers several advantages: it is generic (it can be applied to any metamodel and model independent of the domain), reusable (all tools using the same underlying framework/platform, e.g., all tools based on EMF, can reuse the bridges) and extensible (it can be easily adapted to cover new environments and formats since it addresses separately the syntactic and semantic issues). Besides, our approach is easier to integrate with current trends towards the use of modeling in many aspects of the development process.

The rest of the paper is structured as follows. Next section characterizes the problem context for our method and comments the current state of the art in this field. Section 3 introduces the Eclipse-Modeling example. Then, Section 4 presents our approach for data-level interoperability and applies it to create the Eclipse-DSL Tools bridge. Section 5 repeats the process for the Eclipse-SQL Server Modeling bridge highlighting how both bridges are built following the same generic architecture. Operational-level interoperability is commented in Section 6. Finally, we explain the tool support in Section 7 and the conclusions and further work in Section 8.

2 Problem Definition

The interoperability problem has been widely addressed in the literature (see [23, 19] for existing surveys) but it is still far from being solved. For instance, the OMG has recently created the *Architecture Ecosystem Special Interest Group* to discuss this same problem.

Previous approaches tried to handle this problem by connecting the tools' APIs (e.g., [20]) or interfaces (e.g., [7]). Approaches of this kind, operating at the API-level, may notably make use of the facade pattern. However, this low-level view of tools was too limited to achieve real data interoperability. With the advent of MDE, new proposals have realized about the benefits of looking at the interoperability problem at a higher abstraction level [10] and now follow a model-based approach in which interoperability is specified at the (meta)model level: the internal metamodels of both tools are explicited and aligned and this information is used to drive the interchange of information between them.

Nevertheless, most of these approaches (including our previous experiments in this area) focus on an ad hoc solutions for two concrete tools [16, 9, 18, 21, 15, 24]. The exceptions are [17] that proposes some generic patterns that facilitate a (manual) metamodel alignment based on the use of ontologies (under the assumption that integration of ontology-annotated metamodels is easier) and [8] and [6] that focus on the interoperability of modeling tools through the use of a *bus* that provides several predefined data interchange and conversion services.

Moreover, all these approaches assume that tools have a fixed metamodel (e.g., UML modeling tools only accept models conforming to the UML meta-model). This is not the case anymore. With the rise of MDE, more and more development tasks involve manipulating models conforming to different meta-models and created using generic tools able to handle several metamodels at the same time. Typical examples are the Eclipse and Microsoft modeling tools. As part of the definition of the working environment, the designer can define the metamodel to work with and then create models conforming to that metamodel. Therefore, data interchange for these tools involves bridging both the models and the metamodels at the same time.

In this sense, our approach provides a more general solution to the tool interoperability problem by allowing data interchange between tools with variable metamodels. Once the bridge has been built, metamodels and models can be

automatically interchanged between the tools. Adding new metamodels does not require extending the bridge. Besides, as we will see our approch is fully model-driven and separates in different steps the processing of the syntactic and semantic aspects of the bridge. Instead many existing approaches mix both transformations which impairs the reusability of the bridge. In addition to this, many of the interoperability scenarios cited above could be expressed as a specific instance of our approach (where the variable metamodels would be just the specific metamodel of the tool) and benefit from (parts of) it.

3 Motivating Example

As a concrete example of the problem previously detailed in Section 2, we consider in this paper bridging the *Eclipse Modeling Framework* (EMF) [3] with two different Microsoft modeling environments: *Microsoft DSL Tools* [4] and *Microsoft SQL Server Modeling* [5]. This is actually a quite common interoperability scenario: these three modeling environments overlap in many aspects, in terms of both concepts and capabilities, and are becoming increasingly popular. Therefore, it is likely that many projects need to import/export metamodels (i.e., metadata) and corresponding models (i.e., raw data) from one environment to the other. This can occur for instance when the base platform has to be changed and the related legacy (meta)models must be reused. A collaborative work, in which both environments are being used at the same time and some specified models need to be merged accordingly, is another potential situation where such interoperability is required.

More pragmatically, the goal of our bridges is to allow metamodels and models built or generated in EMF to be manipulated in both Microsoft modeling environments and vice-versa. We provide here short descriptions of these three different environments.

The *Eclipse Modeling Framework* [3] is the well-known reference modeling infrastructure when developing under and for the Eclipse platform. It provides an explicit metametamodel, named *Ecore*, as well as the corresponding standard runtime, serialization and code generation features for the designed metamodels and models to be exploited. See, for instance, the *PetriNet* metamodel along with a sample model conforming to that metamodel created using EMF (cf. Fig. 1). All the Eclipse modeling tools are based on EMF such as model-to-model (M2M) transformation tools (e.g., ATL [14] used to implement the bridges), model-to-text (M2T) transformation tools, graphical or textual model editors, etc.

The *Microsoft DSL Tools* [4] are part of the Visual Studio SDK dedicated to the customization of the Visual Studio platform (largely based on the .NET framework) for specific needs or domains. DSL Tools aim more particularly at providing facilities for building graphical Domain-Specific Languages (DSLs) and corresponding editors, i.e., modeling tools. Contrary to EMF, this environment is based on an implicit metametamodel which is somehow internally hard-coded

by APIs and corresponding serialization XML Schema. It also comes with code generation capabilities from the designed models.

Microsoft SQL Server Modeling (SSM) [5], formerly "Oslo", is the latest modeling environment developed by Microsoft and targets the building of data-driven tools. This environment is based on the 'M' modeling language whose *MSchema* part is a declarative language to design domain models (or meta-models). The other parts of this language allow defining corresponding textual concrete syntaxes for DSLs as well as concrete data models (models). SSM also features a customizable tool, named "Quadrant", allowing to interact between the available models and the actual data (i.e., the databases).

In the remainder of this paper, we will focus on the possible bridges between EMF and these two Microsoft environments. Bridging the Eclipse and Microsoft worlds opens the door to import/export into/from Microsoft all (meta)models specified with any modeling tool built on top of EMF: interoperability is thus possible between many different tools at the same time. We will see the results of porting our PetriNet metamodel and sample model as an example of the application of our method.
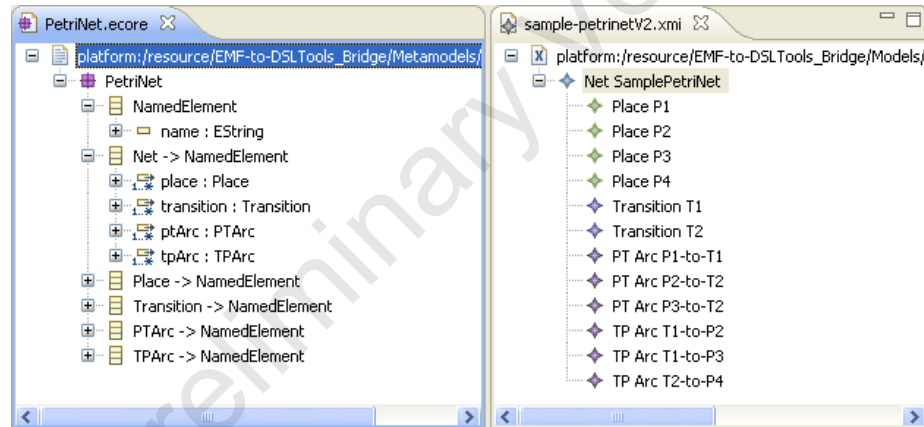

Fig. 1. Simple PetriNet metamodel in Ecore and corresponding sample model (EMF)

4 Approach Presentation

This section introduces our model-driven approach for tool interoperability. First, we present the high-level architecture of the method. Then, we clarify and describe in detail each individual step, showing how to apply the method to build the bridge between *Eclipse Modeling Framework* and *Microsoft DSL Tools*.

4.1 Overview

Fig. 2 depicts a bridge to manipulate within Tool B data (i.e., models) created with Tool A, or vice-versa. Both considered tools are built upon variable metamodel environments. Therefore, each of these environments defines a metametamodel: metametamodel A used by Tool A, and metametamodel B used by Tool B.

With each tool, a given metamodel (e.g., MM_X on the figure) may be expressed in terms of the metametamodel of that particular environment. Then, models (e.g., $M1$) may be expressed in terms of MM_X. The objective of the bridge is therefore twofold:

1. At metamodel-level, the bridge must enable the transformation of any metamodel conforming to metametamodel A into an equivalent metamodel conforming to metametamodel B, and vice versa. For instance, if MM_X is initially expressed in terms of metametamodel A, then the bridge must automatically create the version of MM_X that conforms to metametamodel B.
2. At model-level, the bridge must enable the version of any model defined with Tool A into an equivalent model defined with Tool B, and vice versa. The metamodel of the original model and that of its derived equivalent are themselves equivalent.

The bridge is bidirectional and allows the interchange of models and metamodels in both directions. However, the implementation of the bridge itself must take place inside one of the two environments (in Fig. 2, this bridge is implemented using the environment A)[1]. The main selection criteria is that the selected environment must provide a transformation technology to perform the required adaptations. However, the capabilities of the bridge are independent from the chosen implementation environment.

As seen in Fig. 2, there are four main steps (plus an optional one) involved in the process of creating such a bridge. Each step is represented as a circled number. These steps are:

① **(Optional) Metametamodel discovery.** All variable-metamodel tools necessarily have a metametamodel, in terms of which metamodels are defined. However, this metametamodel may not be explicitly available. In such a case, it is necessary to *discover* that metametamodel from the tool API, its storage schema, etc. For instance, metamodels defined with the tool can be analyzed in order to identify the set of concepts and relationships used to express them. These constitute the metametamodel of the tool. When the metametamodel is readily available (e.g., as is the case for EMF and SSM), this step can be skipped.

---

[1] We could also use a third environment as a pivot but presenting the approach in that way adds unnecessary complexity.
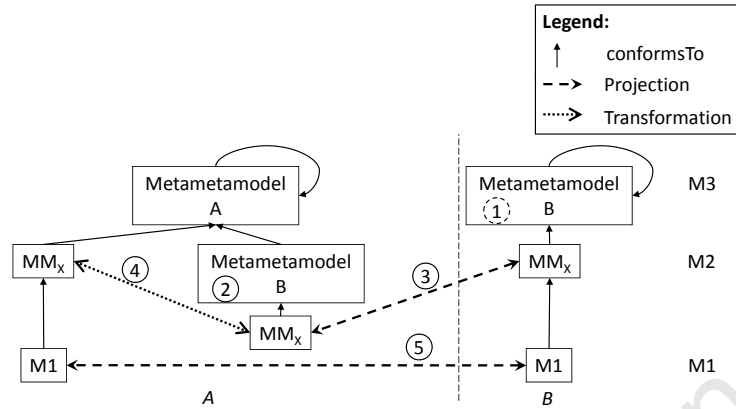
Fig. 2. General bridging approach (overview).

② **Transcription.** This step consists in expressing metametamodel B in terms of metametamodel A (in environment A). This has first to be done manually. However, as a metametamodel conforms to itself, the approach may be bootstrapped and this may be re-generated using the bridge once established.

③ **Syntactic translation.** At this step, the syntactic differences between Tool A and Tool B are solved by transcribing the elements of B within the same technical space of A, i.e., by switching from environment B to A in order to use the same kind of concrete syntax. As seen in Fig. 2, the metamodel in Tool B is re-expressed as an instance of the metametamodel B rewritten within Tool A. Therefore, this is a purely syntactic re-expression we call *projection*, since the structure of elements of B has not changed.

④ **Semantic alignment.** At this step, we cannot yet import models of Tool B in Tool A (and vice-versa) since we cannot have more modeling levels in A, according to the OMG metamodeling architecture. Therefore, we need first to express the metamodel of Tool B as a native metamodel in environment A, which conforms to the corresponding metametamodel of A. This implies a semantic adaptation between the metametamodels of A and B, which is actually realized by *transformations*. These transformations allow to import/export any metamodel between A and B. This step may be realized with the assistance of matching tools or not.

⑤ **Data interchange.** Once this is done, the previous (semantic alignment) information is used as well to generate the *transformations* that actually imports models from B to A and vice-versa. Note that complementary *projections* similar to those of the *Syntactic Translation* step are also required to allow exchanging models between A and B. This bridge is generic: even if Tool B changes its metamodel, there is no need to modify the bridge since the mappings will automatically support importing B models (with the new metamodel) into Tool A.

The important characteristics behind the proposed approach are its genericity, extensibility and reusability:

- **Genericity** because it can be applied on any metamodel and model, independently of the selected environment and considered domain or field of application;
- **Extensibility** because the built transformations and projections can be directly extended in order to target other environments or any software in general, specially given the separation of the syntactic and semantic alignment steps.
- **Reusability** because 1 - these transformations, projections (or at least parts of them) and metamodels can be directly reused as they are for other purposes and 2 - the bridge can be reused by all tools based on the same metametamodel.

Fig. 3 shows the application of this method to our motivating EMF-DSL Tools interoperability example. In this case, the environment A is EMF and the environment B is DSL Tools. EMF has been chosen as the implementation environment because of the several evolved transformation technologies available, such as ATL [14] for model-to-model transformation. Fig. 3 also shows that EMF has an explicit metametamodel named *Ecore* while DSL Tools has an implicit one we arbitrarly name *DSLMeta*. As an example, consider a PetriNet modeling tool in EMF (Tool A) and its equivalent in DSL Tools (Tool B). Each of these two tools is based on an explicit PetriNet metamodel, which conforms to its corresponding metametamodel. The goal is to be able to automatically exchange PetriNet models between the PetriNet EMF modeling tool and the similar DSL Tools one.
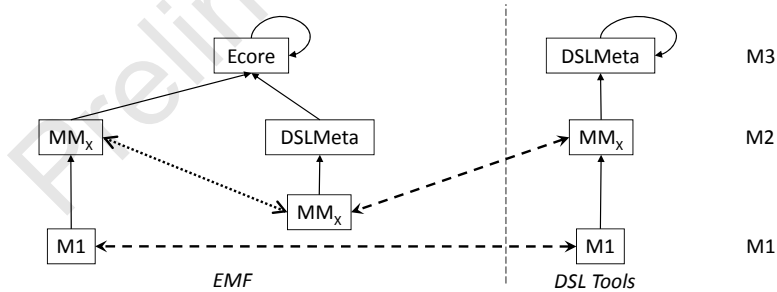

Fig. 3. EMF-DSL Tools conceptual bridge (overview).

In the next subsections, we provide more details on the step-by-step application of our MDE approach to this concrete example. As we will see, for some steps, it is useful to split them into substeps that improve the modularity of our approach and reduce the complexity of each single step. This depends on the

syntactic and semantic distance between the tools to be bridged and it is optional since it is always possible to built the bridge in just the five steps described above.

4.2 Metametamodel Discovery

In our scenario, this optional step is required as the metametamodel of DSL Tools is not explicitly specified. There are currently no fully automated solutions for discovering it. Thus, this has to be performed manually by using the metamodel examples we can find, but also the available documentation and APIs.

4.3 Transcription

In our case, this step requires defining metametamodel *DSLMeta* as a metamodel which conforms to metametamodel *Ecore* in EMF, as shown on Fig. 3. This is a manual step but usually a simple one since many metametamodels share the same basic conceptual elements.
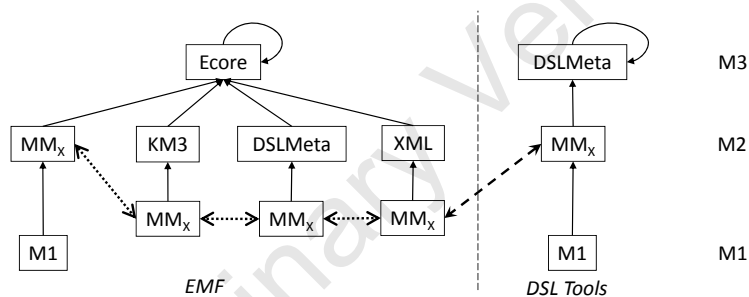

Fig. 4. EMF-DSL Tools metamodel-level bridge (overview).

4.4 Syntactic Translation

Then, we need to be able to express our PetriNet metamodel from DSL Tools as a model conforming to this newly defined *DSLMeta* metamodel in EMF. As shown in Fig. 4 (right), this has been implemented using an intermediate substep to simplify the process. Because the serialization format used by the DSL Tools is XML, we can first automatically *inject* the content of the XML document storing the PetriNet metamodel (in DSL Tools) into a model which conforms to a standard structural *XML* metamodel (note that the inverse operation is of course also possible). At this point, our DSL metamodel is already expressed as a EMF model but conforming to the XML metamodel. Therefore, the second step is to define the model transformation that generates the corresponding version of the model that conforms to the DSLMeta metametamodel in EMF created in the previous step. Using XML as a pivot metamodel simplifies the projection of the PetriNet metamodel in EMF.

4.5 Semantic Alignment

The previously projected PetriNet metamodel can be considered as the precise representation of the initial PetriNet metamodel in DSL Tools. However, this metamodel is not yet conforming to Ecore, i.e., it is not a real metamodel from an EMF point of view (in fact, for EMF, this metamodel is regarded as a simple terminal model, an instance of the DSL Tools EMF metamodel) and cannot be used by metamodeling tools using EMF. The objective of this step is to be able to get a native PetriNet metamodel in *Ecore* from this model and vice-versa, as shown in Fig. 4 (left).

Again, this step uses an intermediate representation to reduce the semantic gap. First, this *DSLMeta* PetriNet metamodel is transformed into a model which conforms to the *KM3* metamodel. Aligning DSL Tools and KM3 is easier than directly aligning DSL Tools and EMF. Furthermore, we already have existing KM3-Ecore converters that take the KM3 model and re-expresses it as a native Ecore metamodel, and vice versa.

At the end of this step, we have a metamodel-level bridge that may now be automatically reused to any metamodel specified in DSL Tools in EMF and the other way round.

4.6 Data Interchange

Now that we have our PetriNet metamodel available in both the EMF and DSL Tools environments, we want the associated PetriNet modeling tools to be able to interoperate exchanging PetriNet models. Fig. 5 presents how this has been concretely realized.
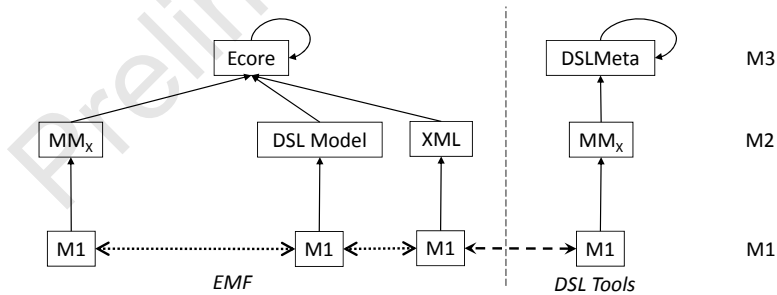

Fig. 5. EMF-DSL Tools model-level bridge (overview).

Similarily to the metamodel-level bridge (i.e., because the serialization format used by the DSL Tools is XML at both levels), the PetriNet sample models in DSL Tools are converted as first *XML* and then *DSLModel* models in EMF and vice-versa. The *DSLModel* metamodel is introduced in order to decouple

concrete syntax (i.e., *XML*), and metamodel-independant abstract syntax. This metamodel represents the graph structure used in DSL Tools independently of any metamodel. This corresponds to the *projection* phase between the two different environments.

The *transformation* phase itself is separated into two distinct parts, which makes it fully generic (i.e., independent from the used metamodel). First, the transformation itself is (semi)automatically generated from the alignment information used in the previous step. Then, the transformation is added to the overall transformation chain for effectively building the output terminal model from the source one. Only parts of the transformation that are metamodel-dependant are automatically generated. Metamodel-independant parts are written by a developer once for each bridge. They are then reused for every metamodel to which the bridge is applied. Generated and manually written transformation parts are typically composed by chaining them.

This way, we can apply the generated PetriNet-DSLModel mapping transformations in order to finalize the bridge and interchange PetriNet models (coming from either EMF or DSL Tools).

In this section, our generic MDE approach for tool interoperability has been introduced and directly used on our first motivating example. The next section demonstrates the genericity and applicability of our solution by considering a second example: bridging EMF and SQL Server Modeling.

5   Bridging Eclipse and SQL Server Modeling

The generic interoperability method presented in the previous section (cf. Section 4) can be applied to make interoperate many different platforms and their corresponding tools. As a second example, we briefly describe in this section how we can use our method to build an *EMF-SQL Server Modeling* bridge. To do so, we follow again the same steps we considered for the creation of the *EMF-DSL Tools* bridge.
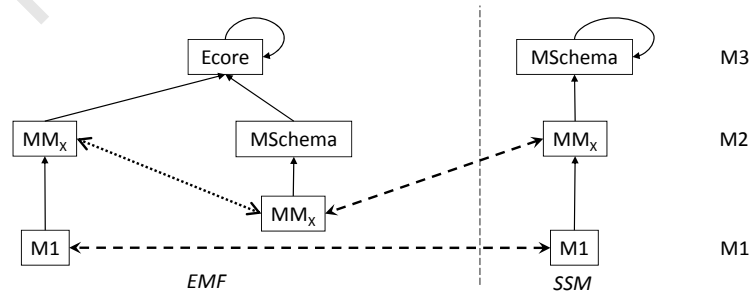

Fig. 6. EMF-SSM theoretical bridge (overview).

Fig. 6 presents the abstract view of this bridge in this specific case. We consider here *MSchema*, i.e., the part of the "M" language dedicated to meta-modeling (cf. Section 3), as the SSM metametamodel. As the situation is roughly equivalent to the *EMF-DSL Tools* bridge one, as shown from Fig. 3, we do not provide more insights on this overall view.

Fig. 7 gives more concrete details on the metamodel-level bridge. Again, the architecture is roughly the same as in the *EMF-DSL Tools* bridge: *KM3* is used as a pivot metamodel which allows directly reusing the available Ecore-KM3 converters, while the actual mapping between the two metametamodels is realized by the *KM3-to-MSchema* transformation. The only difference is that, for this bridge, we are not using XML for metamodel/model serialization since the format used by SSM is not XML-based but text-based. The use of the *XML* metamodel as an intermediate step is thus not required: a textual modeling tool can be directly applied to switch between a textual file and the corresponding metamodel/model, and vice versa. For this purpose, we use the TCS [13] (Textual Concrete Syntax) tool.
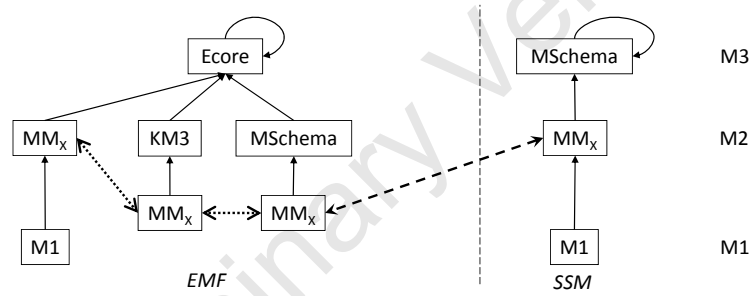

Fig. 7. EMF-SSM metamodel-level bridge (overview).

6 Operational-Level Interoperability

So far, we have focused on the data-interoperability problem. The bridges presented in the previous section enable sharing models among the tools, including the interchange of transformation models, i.e., models that define model transformations between source and target (meta)models. However, the simple exchange of a transformation model is a necessary but not sufficient condition to achieve operational-level interoperability. The additional requirement is that the target tool includes a transformation engine able to process the information contained in the transformation model and execute the corresponding transformation.

In our scenario, operational-level interoperability between Eclipse and Microsoft modeling tools requires creating a new version of the ATL virtual machine (component in charge of executing model-to-model transformations defined us-

ing the well-known ATL transformation language [14]) adapted to the Microsoft modeling tools.

The ATL virtual machine is currently written in Java and only accepts (meta)models defined using EMF, KMF [12] or MDR as modeling frameworks. Therefore, porting the ATL virtual machine implies two different steps:

1. Migrating ATL VM to the .NET platform to facilitate its execution from within the Microsoft tools. It would be possible to directly call the Java ATL VM from .NET but this solution loses in efficiency and elegance, requiring the use of both Java and .NET Framework virtual machines (and exchanging data between them) at the same time.
2. Integrating support for the SSM and DSL Tools frameworks.

The first step has been already completed. Regarding the second step, the virtual machine has been built from the beginning in a layered structure (see Fig. 8) to facilitate its portability. The model adaptation layer decouples the virtual machine's core from the modeling framework used to define the (meta)models. This allows the virtual machine to run on top of new modeling frameworks providing that an adapter for the framework is available. Therefore, adding support for SSM and DSL Tools only requires to create the corresponding adaptors.
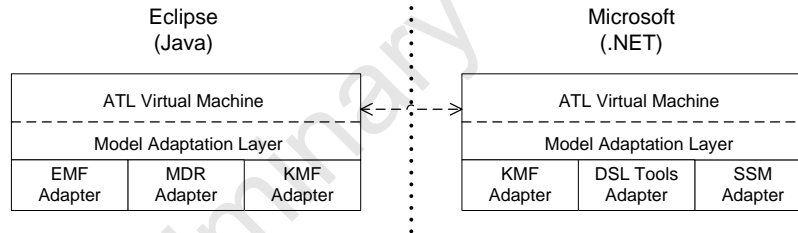

Fig. 8. ATL VM structure (overview).

7   Tool Support

Several of the bridges described in the previous sections have been actually implemented (see Fig. 9) and are available from [2]. As can be seen in the figure, in some cases, bridges concern both the metamodel and model levels, or just one of these two categories. We are working on completing the full set of bridges.

However, we would like to remark that, in fact, it is not necessary to implement all bridges to achieve full interoperability between each pair of tools. Existing bridges can be used, by transitivity, to connect two tools with no direct bridge between them. For instance, even if the corresponding bridge is not implemented, we can interchange metamodels and models between DSL Tools and

SSM using EMF as a pivot tool. This is similar to using a metamodel as a pivot between two other metamodels. However, in this case, we use it for bridges, not single transformations.
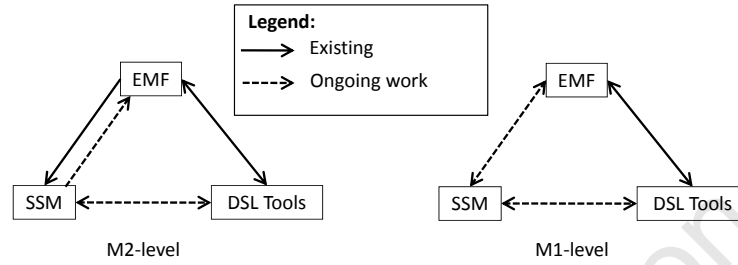

Fig. 9. Existing bridges (overview).

As an example of the use of the bridges, Fig. 10 shows the result of automatically generating, for the DSL Tools, the *metamodel* and sample *model* from our *PetriNet* EMF example (Fig. 1).

Note that this same *PetriNet* metamodel is also available in SSM format. However, the SSM equivalent sample model (in "M") has not been produced since the direct bridge has not been yet implemented at the model-level. But, as commented before, it could be obtained by transtivity of the other bridges.
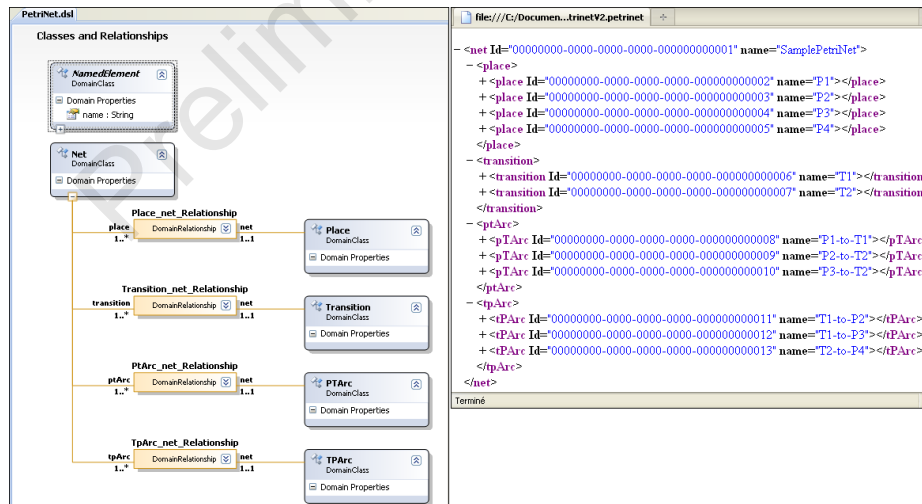

Fig. 10. PetriNet metamodel and sample model in DSL Tools format.

As part of our tool support, we have also released a first version of the ATL Virtual Machine for .NET [1] that allows direct execution of model transformations within the .NET environment.

8   Conclusions and Future Work

We have presented our approach for tool interoperability, focusing on the most general scenario: interoperability among tools able to handle data (i.e., models) conforming to different metadata (i.e., metamodels). Our method follows a model-driven approach in which the (meta)metamodels of the tools are explicited, aligned and used to (semi)automatically generate the model-to-model transformations that effectively bridge the tools.

Our model-driven view of the problem facilitates the reusability, genericity and extensibility of the bridges in order to cope with the increasing complexity of tool ecosystems. This model-driven view is also especially useful under the current model-driven engineering paradigm where most of the tools already use (meta)models as first-class entities.

There are several directions in which we plan to continue this work. First, we plan to improve our proof of concept by completing the implementation of all bridges between Eclipse and Microsoft modeling tools (including a bridge between the two Microsoft tools themselves) and extending them with full support for operational-level interoperability as well. Second, we would like to improve the automation of the interoperability process by advancing, for instance, in the automatic discovery of metamodels from tool APIs for those tools with no explicit metamodel. Finally, we plan to study different "instantiations" of our generic architecture to see how it can be optimized depending on the specific pair of technical spaces (e.g., XML, grammar-based, modeling-based) of the two tools to bridge.

References

1. AmmA .NET. http://www.emn.fr/z-info/atlanmod/index.php/AmmADotNet.
2. Eclipse-Microsoft Bridges Implementations. http://docatlanmod.emn.fr/Eclipse-Microsoft_Bridges/Implementations/.
3. Eclipse Modeling Framework (EMF). http://www.eclipse.org/modeling/emf/.
4. Microsoft Domain-Specific Language (DSL) Tools. http://msdn.microsoft.com/fr-fr/library/bb126235.aspx.
5. Microsoft SQL Server Modeling Technologies. http://msdn.microsoft.com/en-us/data/default.aspx.
6. Open tool Integration Framework. http://www.escherinstitute.org/Plone/frameworks/otif.
7. Y. Bao and E. Horowitz. A new approach to software tool interoperability. In *SAC '96: Proc. of the 1996 ACM Symposium on Applied Computing*, pages 500–509, New York, NY, USA, 1996. ACM.

8. X. Blanc, M.-P. Gervais, and P. Sriplakich. Model bus: Towards the interoperability of modelling tools. In U. Aßmann, M. Aksit, and A. Rensink, editors, *MDAFA*, volume 3599 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2004.

9. M. Didonet Del Fabro, J. Bézivin, and P. Valduriez. Model-driven tool interoperability: An application in bug tracking. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 4275 of *Lecture Notes in Computer Science*, pages 863–881. Springer, 2006.

10. B. Elvester, A. Hahn, A.-J. Berre, and T. Neple. Towards an interoperability framework for model-driven development of software systems. In *Proc. of the 1st Int. Conf. on Interoperability of Enterprise Software and Applications*, pages 409–420, San Diego United States, 2005. Springer.

11. A. Geraci. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries*. Institute of Electrical and Electronics Engineers Inc., The, 1991.

12. F. Jouault, J. Bézivin, and M. Barbero. Towards an advanced model-driven engineering toolbox. *ISSE*, 5(1):5–12, 2009.

13. F. Jouault, J. Bézivin, and I. Kurtev. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In *GPCE'06: Proc. of the 5th Int. Conf. on Generative programming and Component Engineering*, pages 249–254, 2006.

14. F. Jouault and I. Kurtev. Transforming Models with ATL. In *MoDELS Satellite Events*, pages 128–138, 2005.

15. H. Kern and S. Kuhne. Model interchange between aris and eclipse emf. In *7th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA 2007*, 2007.

16. H. Kern and S. Kuhne. Integration of microsoft visio and eclipse modeling framework using m3-level-based bridges. In *Proc. of the 2nd ECMDA Workshop on Model-Driven Tool and Process Integration*, 2009.

17. G. Kramler, G. Kappel, T. Reiter, E. Kapsammer, W. Retschitzegger, and W. Schwinger. Towards a semantic infrastructure supporting model-based tool integration. In *GaMMa '06: Proc. of the 2006 Int. Workshop on Global integrated model management*, pages 43–46, New York, NY, USA, 2006. ACM.

18. N. Moalla, H. Chettaoui, Y. Ouzrout, F. Noel, and A. Bouras. Model-Driven Architecture to enhance interoperability between product applications. In *Int. Conf. on Product Lifecycle Management (PLM08)*, pages –, Séoul Corée, République de, 07 2009.

19. H. Ossher, W. H. Harrison, and P. L. Tarr. Software engineering tools and environments: a roadmap. In *ICSE - Future of SE Track*, pages 261–277, 2000.

20. S. E. Sim. Next generation data interchange: Tool-to-tool application program interfaces. In *WCRE '00: Proc. of the 7th Working Conf. on Reverse Engineering (WCRE'00)*, page 278, Washington, DC, USA, 2000. IEEE Computer Society.

21. Y. Sun, Z. Demirezen, F. Jouault, R. Tairas, and J. Gray. A model engineering approach to tool interoperability. In D. Gasevic, R. Lämmel, and E. V. Wyk, editors, *SLE*, volume 5452 of *LNCS*, pages 178–187. Springer, 2008.

22. I. Thomas and B. A. Nejmeh. Definitions of tool integration for environments. *IEEE Softw.*, 9(2):29–35, 1992.

23. M. N. Wicks and R. G. Dewar. Controversy corner: A new research agenda for tool integration. *J. Syst. Softw.*, 80(9):1569–1585, 2007.

24. T. Zhang, F. Jouault, J. Bézivin, and X. Li. An mde-based method for bridging different design notations. *ISSE*, 4(3):203–213, 2008.