A Practical Approach to Bridging Domain Specific Languages with UML profiles

Anas Abouzahra, Jean Bézivin, Marcos Didonet Del Fabro, Frédéric Jouault

ATLAS Group (INRIA & LINA, University of Nantes)

{Abouzahra.Anas | bezivin | marcos.ddf | f.jouault}@gmail.com

ABSTRACT

This paper considers that there are two important views of Model Driven Engineering. One consists in starting with a well known standard universal modeling language like UML 2.0 and to define restrictions and extensions to this all-purpose language by the way of so-called "profiles". The other possibility consists in using small, well-focused Domain Specific Languages and to deal with the coordination between these. We will not discuss here the pros and cons of both approaches but we will consider that each has many supporters while the debate is going on. UML CASE tools currently produce a significant legacy, mainly of UML profiles. Bridges between both modeling worlds will surely be most needed in the future. We consider this bridging problem to be a hard one and we propose to use some advanced modeling tools to solve it. We show how a combination of a model-transformation tool (ATL) and a model-weaving tool (AMW) may help solving this problem. We also show how to take advantage of higher-order transformations in this endeavor.

Keywords

DSM; DSLs; UML; Profiles;

1. INTRODUCTION

Model Driven Engineering (MDE) has matured to the point where it can offer significant leverage in all aspects of software development. Nowadays basically two approaches appear on the top of this technology: the Unified Modeling Language (UML) [8] and Domain Specific Languages (DSLs). The first one, UML, provides a rich set of modeling notations and semantics, and allows developers to customize it to their particular domain or purpose and to understand, specify, and communicate their application designs at a higher level of abstraction. It provides some extension mechanisms to cover as many domains as possible (stereotypes, tagged definitions and constraints). A consistent set of such extensions is named an UML profile. UML profiles are used to model those aspects of systems or applications that are not directly describable by native UML elements. This approach will be called ***UML-profiling*** through this paper.

Another way to represent applications is to specify each aspect separately, with a dedicated formalism. We call ***DSM*** this approach (Domain Specific Modeling). DSLs can provide a solution to capture, represent and manage these aspectual concepts [4]. A DSL is a language dedicated to a particular domain, problem or task. It may be associated to a given metamodel. It is usually small and well focused. It provides appropriate abstractions and notations to describe a particular domain. This may be contrasted to the first approach of UML-profiling where an all purpose language is used for all kind of modeling actions. It seems that most DSM approaches need a post-integration phase to coordinate the description fragments conforming to various DSLs. This is less necessary in an UML-profiling approach. On the contrary the DSM approach seems to lead more naturally to modularity of expression than the UML-profiling approach.

Whatever the compared advantages and drawbacks of both approaches, they are currently used by increasing communities. The volume of artifacts produced in UML-profiling as well as in DSM-based processes is rapidly increasing. Whatever the future evolution of MDE in either branch, we are accumulating a large legacy of such artifacts. Thus, we need interoperability between these approaches.

This paper identifies more precisely the relation between UML and DSL approaches and proposes a tool that can bridge them. The concept of bridge is used to denote the capability of interoperability between the two spaces. When a corresponding bridge is available for a metamodel and a corresponding profile, it is possible to produce automatically an UML profiled model from a model conforming to the metamodel in question and conversely.

To implement a bridge between UML profiles and DSLs, we use the functionalities provided by AMMA (ATLAS Model Management Architecture) [3], a new generation model-engineering platform. AMMA is built on top of the Eclipse Modeling Framework (EMF) [6] and consists of four different components. The components that play the most important role in the bridge implementation presented in this paper are a model transformation language named ATL (ATLAS Transformation Language) and a tool for representing explicit correspondences between models named AMW (ATLAS Model Weaver) [7].

The implemented tool takes as input an UML profile and the metamodel of the system described by the profile. It allows transforming between models conforming to those inputs. It transforms UML models designed with this profile to models conforming to the profiled metamodel and vice versa. Thus, the tool aims at automatically generating transformations between models and not transforming a metamodel to a profile or the opposite. Figure 1 shows a use case of the tool. *MMa* is a metamodel for which a profile already exists. *Ma.uml* is an UML model designed using the profile for *MMa*. The tool will create *Ma*. *Ma* is the model conforming to *MMa* and corresponds to *Ma.uml*. It also will be able to do the opposite transformation: creating *Ma.uml* from *Ma*.

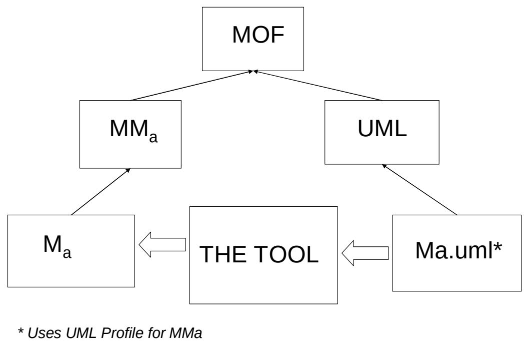


Figure 1 A tool use case

The bridge spans two levels: The first between an UML profile and a metamodel and the second between UML models and models conforming to the metamodel. An AMW weaving model is used to implement the first level then two ATL-based transformations enable the automatic generation of two ATL models that constitute the second. The benefit of using such a bridge is the ability to transpose UML models designed with a UML profile to a DSL platform, and conversely.

The lessons learnt in the realization of the bridge between UML profiles and DSL are very helpful in using both AMW and ATL for building other exchange schemes between platforms. The architecture we have implemented, an AMW model to implement mapping links and ATL transformations using this model to produce bridges, may be adaptable to other technical spaces bridging.

This paper is organized as follows. In section 2 we describe the two modeling technologies bridged: DSM and UML profiling. Section 3 presents the AMMA platform used in this work and our implementation of the tool. Section 4 explains problems and limitations of our actual implementation. Section 5 concludes the paper.

2. UML Profiling and DSM

DSM and UML-profiling represent the two main modeling points of view: UML used as a general-purpose language that is aimed at any software problems and DSLs that aim at representing each domain with a specific metamodel. This section provides a brief introduction to DSLs and UML profiles.

2.1 DSM

A DSL is a computer language dedicated to a particular kind of problem. As a simplification we consider here that a DSL is based on a specific metamodel even if a more complete definition would be necessary, but falls beyond the scope of this paper. A metamodel contains a set of concepts, relations between these concepts and constraints. Metamodels describe the structure of models. A model conforms to a metamodel. The Meta Object Facility (MOF) [10] is a language to define metamodels; it is thus a metametamodel. The MOF is self-defined. Figure 2 shows the classical OMG organization of models with MOF on top of the hierarchy and UML at level M2.

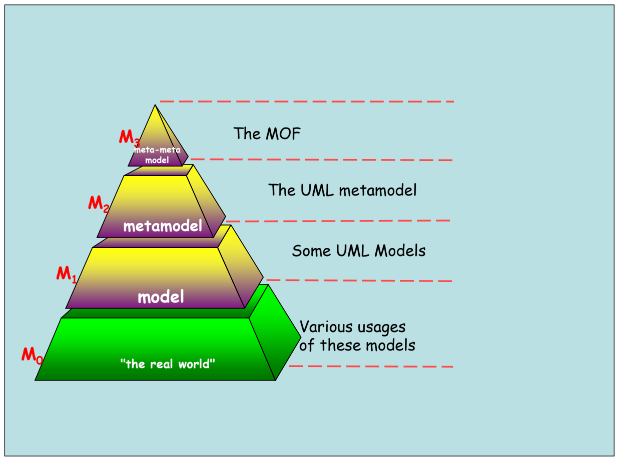


Figure 2 Hierarchy of models

DSLs [1] are usually small and declarative, offering only a restricted suite of notations and abstractions for specifying a particular domain. We can consider that level M2, as mentioned in Figure 2, corresponds to a vast library of MOF-compliant metamodels. The DSM approach does not contradict the OMG organization.

The DSM approach has been recently pioneered by Microsoft [14] and initial tool support is being developed in DSL Tools.

2.2 UML Profiles

When the UML language is not able to represent a system or an application in a convenient way, a UML profile may provide additional devices to do it [2]. A profile defines virtual UML subclasses by associating stereotypes, tag definitions and constraints to provide some additional meaning to UML basic elements.

A stereotype provides a way of defining virtual subclasses of UML metaclasses with additional semantics. It may define tags with additional properties as well as additional constraints over its base metamodel class. The actual properties of individual model elements are specified using tagged values.

MOF Element	**UML Element**
Package	Model or Package, both with <<metamodel>> stereotype
Association	Association
Exception	Exception or Class with <<exception>> stereotype
Attribute	Attribute
Class	Class

Table 1 Excerpt of the profile for MOF

A constraint is a semantic restriction represented as a text expression. It can be specified using the Object Constraint Language (OCL) [13].

Table 1 presents a part of the profile for MOF, an UML profile that is used as an example throughout the paper [9].

Each profile element maps to a specific MOF element as shown in the table above. Mapping details for each element completes this mapping table. They define more precisely each map between UML and MOF elements. Each mapping detail contains subsections covering these topics: tags, mapping properties, constraints, and limitations.

Tags are used for MOF properties not directly supported by UML. However, some MOF details cannot be rendered in UML using this profile. Those details are described as limitations. The tables below show examples of mapping details of the UML Model to the MOF Package.

Tag	**Value**
org.omg.uml2mof.hasImplicitReferences	true (default) or false

Table 2 Excerpt of UML Model tagged values

MOF Feature	**UML**
container	namespace or null if the namespace is either null or not mapped to a MOF Package
contents	ownedElement, taggedValue
isRoot	isRoot
isLeaf	isLeaf
isAbstract	isAbstract

Table 3 Excerpt of UML Model feature table

Limitations
The order of elements is not fully preserved when rendered using the profile as UML since it has separate associations for *ownedElement* and *taggedValue*

Table 4 UML Model Limitations

Constraints
UML Model/Package representing a nested MOF Package must not have a tag *org.omg.uml2mof.hasImplicitReferences.*

Table 5 UML Model Constraints

It should be noticed that there is no normalized way to define UML profiles. However, most of them are or can be specified using a profile definition table like Table 1 for the profile for MOF accompanied with mappings details to explain each profile definition.

The following excerpts from two OMG UML profiles specifications illustrate this point.

UML Profile for CORBA Components [11]: Component profile definition

A CORBA Component is defined using a UML "*CORBAComponent*" stereotyped class. A "*CORBAComponent*" can inherit from another one (single inheritance) using the UML generalization. It can also inherit from a set of CORBA interfaces. These relationships are made concrete with "*CORBASupports*" stereotyped generalizations.

Stereotype	**Base Class**	**Parent**	**Tags**
CORBAComponent <<CORBAComponent>>	Class	CORBA Interface	NA
CORBASupports <<CORBASupports>>	Generalization	NA	NA

Table 6 CORBA Component profile definition

Descriptions:

- A *CORBAComponent* is a class with specific, named collection of features like attributes or ports. An instance of the component has state and identity.
- *CORBASupports* is a generalization relationship between component and its inherit interface(s).

UML Profile for Java and EJB [12]: Java profile

The convention used in this profile is that the classes from the Java metamodel are expressed as stereotypes in a Java model for use primarily in UML class diagrams.

The attributes of the Java metamodel classes are tags to be applied on elements bearing the class stereotypes. The tag names are qualified by the stereotype to which they are applied, since it is possible for a UML element to bear more than one stereotype.

Metamodel element name	**Stereotype name**	**UML base Class**	**Tags**
JavaClass	<<JavaClass>>	Class	JavaClass.isPublic boolean JavaClass.isAbstract boolean JavaClass.isFinal boolean
JavaPackage	<< JavaPackage>>	Class	
ArrayType	<< ArrayType>>	Class	ArrayType.arrayDimentions Integer

Table 7 Extract of UML profile for Java

OMG UML profiles formal specifications are defined with profile definition tables like above and are completed by descriptions, constraints, mapping details, etc. They don't use exactly the same format but they treat the same information.

3. An Implementation using AMMA Platform Tools

AMMA extends the EMF facilities in a number of ways. By design, AMMA integrates the notion of technical space, i.e. the possibility to interoperate with other MDE and non-MDE environments. This section provides a brief presentation of AMMA tools that we use to implement the bridging tool then it explains the implementation details.

3.1 AMMA

Currently built on top of EMF, AMMA [3] brings additional functionalities and could be implemented on top of other MDE environments. Most of the facilities developed in this project are or will be available as open source on the Eclipse GMT project (www.eclipse.org/gmt). AMMA has both local and distributed implementations and is based on four blocks providing a set of model processing facilities:

- Atlas Transformation Language (ATL) defines model transformation facilities for a QVT-like language: a transformation virtual machine, a metamodel-based compiler, a debugging environment etc.
- Atlas ModelWeaver (AMW) makes it possible to establish links between the elements of two (or more) different models.
- Atlas MegaModel Management (AM3) defines the way the metadata is managed in AMMA (global distributed registry on the models, metamodels, tools etc.)
- Atlas Technical Projectors (ATP) defines a set of injectors/extractors enabling to import/export models from/to foreign technical spaces (Java classes, relational models, etc.)

3.2 Global Overview of Bridging

Before explaining the implementation of the bridge between UML profiles and DSLs we give the overall idea behind the project. Figure 3 provides an overview of the approach.

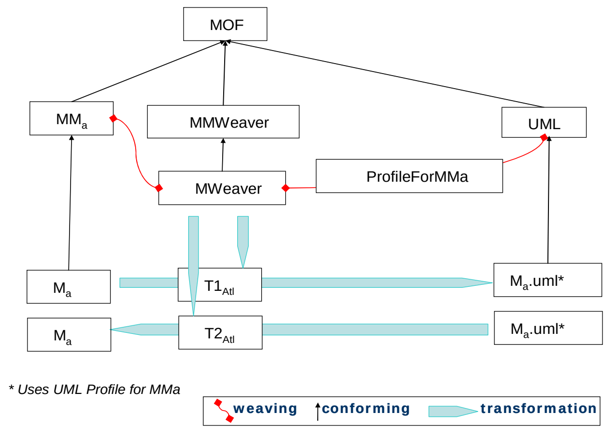


Figure 3 Global overview of the bridging

To make a complete bridge between an UML profile (*ProfileForMMa*) and a DSL metamodel (*MMa*) we must establish the mappings between their elements. We use an AMW model (*MWeaver*) to this aim. AMW is an efficient tool for explicitly representing correspondences between models. The correspondences are stored in a weaving model conforming to a weaving metamodel (*MMWeaver*). This allows us, using two ATL transformations (vertical arrows in Figure 3), to generate two other transformations: T1 and T2. T1 transforms models conforming to *MMa* into profiled UML models. T2 transforms profiled UML models to models conforming to *MMa*.

3.3 Implementation Choices

Before implementing the tools we have been obliged to make some decisions about the way to proceed. As shown in Figure 3, we can use UML to specify profiles. However, this may have some drawbacks:

- The UML modeling tools use several versions of UML (UML2.0, UML1.4, etc.) which are not necessary compatible.
- The Model Weaver tool (AMW) is only compatible with EMF Ecore (The Eclipse/EMF metametamodel close to MOF 2.0) models.
- The weaving process has to load the entire UML metamodel each time we specify a new bridging with a new UML profile, even if just some of its elements are needed.

For those reasons we decided to design a profile metamodel. This way, there is the possibility to specify each profile separately as a model with the required elements. We also resolve the problem of UML versions. The new configuration of the tool is described in Figure 4.

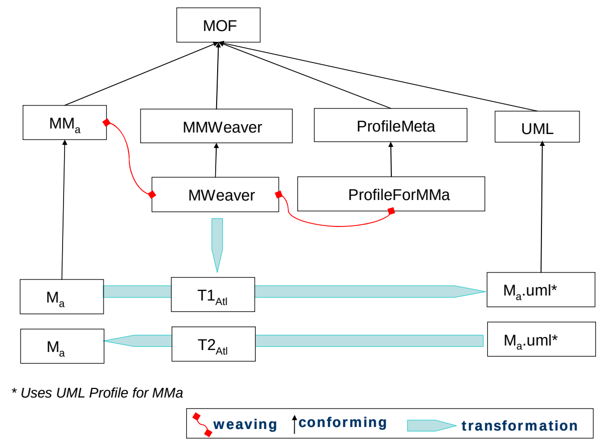


Figure 4 The tool new configuration

Mapping details specify links between UML elements and the elements from the profiled metamodel. This information should not to be redundant even if the transformation is bidirectional. This requires finding a way to represent it separately of the two input models (the profile model and the profiled metamodel). A separate representation of mapping links removes redundancy and facilitates the implementation of a bi-directional transformation because it can be obtained from the same representation.

Modeling mappings details require also some structures. We need to express conditional mapping, design new types, define OCL expressions, and navigate both elements of profiles and profiled metamodels. We illustrate some examples below. Table 8 shows the mappings details for the MOF *AssociationEnd* that is profiled by an UML *AssociationEnd*.

MOF Feature	**UML**
type	participant
multiplicity	*lower* and *upper* are determined by *multiplicity.range*, *isOrdered* is mapped to *ordering* (where true corresponds to ordered) and *isUnique* maps always to true
aggregation	*aggregation* (UML aggregate matches MOF shared)
isNavigable	*isNavigable*
isChangeable	*changeability* (changeable maps to true)

Table 8 UML AssociationEnd feature table

Constraints
An association must have exactly two ends. UML changeability must be either changeable or frozen. Multiplicity must have a single range.

Table 9 UML AssociationEnd constraints

The attribute "multiplicity" is an example of the construction of a new type since it does not correspond exactly to an UML element. It is designed from several different elements as shown in Figure 6 below. The attribute "*isOrdered*" is an example of a conditional mapping.

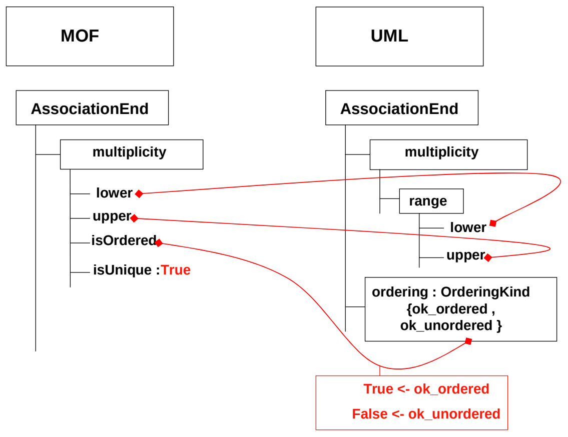


Figure 5 AssociationEnd mapping

Table 9 shows an example of constraints, which can be described with OCL expressions.

At this point of the project description, we may comment on the importance of the model-weaving tool AMW. The intuition of building this tool was that the QVT-like model transformation languages like ATL were necessary but insufficient to solve practical problems. We have many other application fields for AMW, but this is the first time we encountered a situation that could not be solved efficiently without this tool. AMW is necessary to implement mapping links. Each kind of mapping is defined with a specific kind of weaving correspondence. Section 3.5 explains more precisely the AMW implementation of correspondences.

3.4 The Profile Metamodel

To define a Profile metamodel we have to take care of some points:

- The definition of each virtual UML subclass, which corresponds to a definition of a Profile element (in particular a class of the profiled metamodel), can be done in several ways. In fact, each element of the profiled metamodel may be represented with more than one UML element. We may represent a Package by an UML Model or an UML Package in our example (Profile for MOF).
- A UML element can take more than one stereotype to define the same element of the profiled metamodel. However, it can also take different stereotypes to define different elements.
- We should have the possibility to navigate the UML elements used by the Profile and have access to some of their attributes and values.

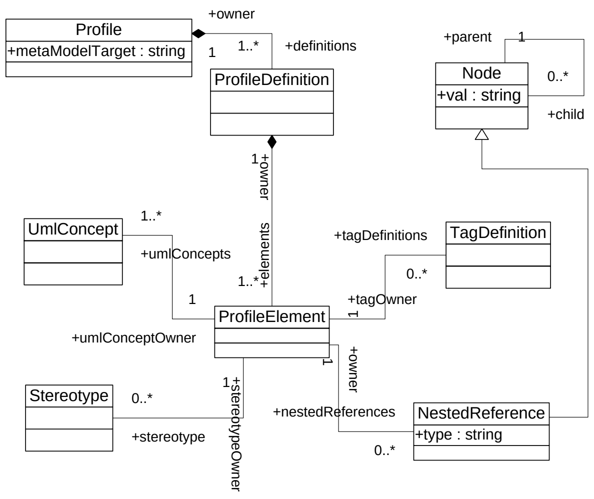


Figure 6 Simplified version of the profile metamodel

Figure 6 shows a simplified version of the profile metamodel we have designed for the tool. In this metamodel a *ProfileDefinition* corresponds to the definition of a metamodel element using the profile. As we said, this definition can be done in several ways, so a *ProfileElement* describes each one. A *ProfileElement* contains one or more *UmlConcept* which specify the UML elements associated to the profile element, stereotypes, and the *NestedReferences* needed to define mapping links. Nodes allow the possibility to navigate the UML elements.

UML profiles may thus be defined as models conforming to this metamodel as illustrated by Figure 4.

3.5 Implementation of Mapping Links with the AMW Tool

The weaving model takes the profile model as its left input and the profiled metamodel as its right one. We have defined five kinds of correspondences:

- **LinkDef**: it defines a link between the *ProfileDefinition* and the metamodel element refers to. It has a left element, a right element and children which can be of type *LinkElem*.
- **LinkElem**: it defines a link between the *ProfileElement* and the metamodel element refers to. It has a left element, a right element and children, which can be of type: *Nested*, *IfNested* or *NewType*.
- **Nested**: it defines direct mapping between UML elements attributes and metamodel elements ones. It has just a left element and a right one.
- **IfNested**: it is a kind of mapping table. We use it when metamodel element attributes do not correspond exactly to UML element ones. It has a left element, a right element and *Nested* links, *IfNested* links and *NewType* links which are needed to define mappings.
- **NewType**: We use this kind of correspondence to define a metamodel element with UML attributes, which pertain to different UML elements. It has a right element and *Nested* links, *IfNested* links and *NewType* links which are needed to define mappings.

Let us take the example of Figure 5 to show a use case of the correspondences defined above. The corresponding weaving is illustrated by Figure 7.

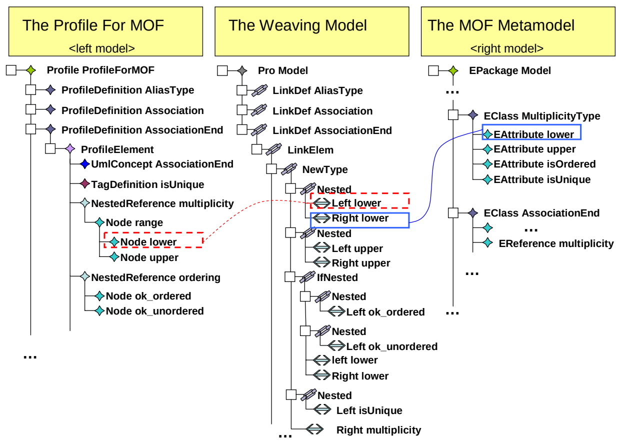


Figure 7 A weaving example

As shown in Figure 7, left and right children of weaving link structures are loaded from elements of left and right models. For example, the left child "lower" of the first *Nested* link of the *NewType* link corresponds to the Node "lower" in the left model. The right child "lower" of this *Nested* corresponds to the attribute "lower" of the class *MultiplicityType* in the right model.

3.6 ATL Transformations

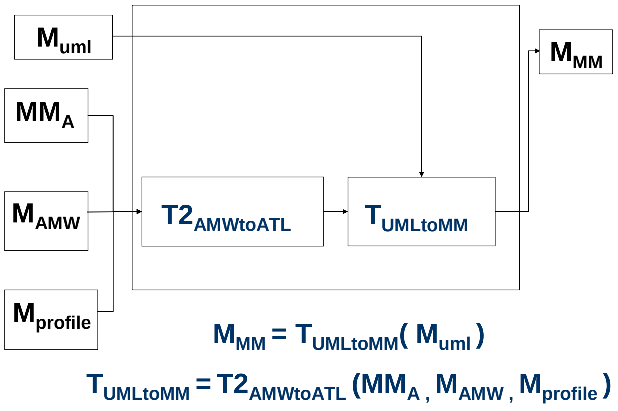
The tool uses two ATL transformations: $T1_{AMWtoATL}$ and $T2_{AMWtoATL}$ to automatically generate the two metamodel-specific ATL programs. As shown in the figures Figure 8 and Figure 9, $T1_{AMWtoATL}$ and $T2_{AMWtoATL}$ take as input the profiled metamodel, the profile model and the weaving model. The first output transformation is $T_{MMtoUML}$, which transforms UML profiled models to a model conforming to the profiled metamodel. The second transformation output is $T_{UMLtoMM}$, which transforms models conforming to the profiled metamodel to UML profiled models.

We provide below some of the major characteristics of the transformation rules for the transformations $T1_{AMWtoATL}$ and $T2_{AMWtoATL}$. The whole source code of this work is available under the GMT open source project.

- The ATL model has to be created from the AMW model:
  - Its input model has to be created from the left model of the weaving for $T1_{AMWtoATL}$ and have to correspond to a simplified UML metamodel only with the elements needed for the bridging (the elements present in the profile model). Its output model has to be created from the right model and have to correspond to the profiled metamodel MM_A and vice versa for $T2_{AMWtoATL}$.
- For each *ProfileElement*, a matched rule has to be created:
  - The input pattern has to contain the right or the left element of the *LinkElem* link according to the

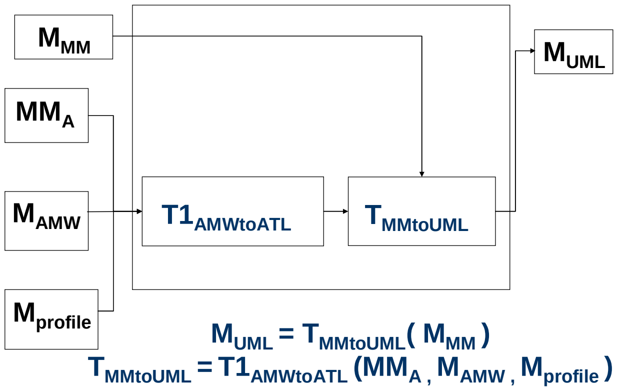
transformation we are implementing ($T1_{AMWtoATL}$ or $T2_{AMWtoATL}$).

- The output patterns have to contain, in addition of the right or the left element of the *LinkElem*, other output patterns necessary for the matched rule: from *NewType* links, *TagDefinitions*, etc.
- The bindings have to be created from all the children of the *LinkElem* referring to the *ProfileDefinition* which outputs the matched rule: *Nested* links, *IfNested* links, *NewType* links, etc.



M_{uml}
MM_A
M_{AMW}
$M_{profile}$
$T2_{AMWtoATL}$
$T_{UMLtoMM}$
M_{MM}
$M_{MM} = T_{UMLtoMM}(M_{uml})$
$T_{UMLtoMM} = T2_{AMWtoATL} (MM_A , M_{AMW} , M_{profile})$

Figure 8 Transformation from UML profiled models to a model conforming to the profiled metamodel



M_{MM}
MM_A
M_{AMW}
$M_{profile}$
$T1_{AMWtoATL}$
$T_{MMtoUML}$
M_{UML}
$M_{UML} = T_{MMtoUML}(M_{MM})$
$T_{MMtoUML} = T1_{AMWtoATL} (MM_A , M_{AMW} , M_{profile})$

Figure 9 Transformation from models conforming to the profiled metamodel to UML profiled models.

4. Limitations

The correspondence structures and the profile metamodel we have defined are still probably insufficient to describe all profiles and their mapping with metamodels. In fact, as we have explained in section 2.2, there is no normalized way to define UML profiles and their mappings with MOF metamodels although the OMG is standardizing some of them. However, the goal of this work is to convince about the possible bridging between DSLs and UML profiles. What was done till now shows the possible bridging of a large set of UML profiles. A deeper analysis of UML profiles is necessary to consider additional structures for complementing both the profile and the weaving metamodels.

Some initial tests we have made with the implemented tool already show a few limitations that may be solved later:

For T2AMWtoATL:

- Our implementation allows associating one stereotype by ProfileElement, whereas an UML profile can map a metamodel element to an UML one with more than one stereotype.

For T1AMWtoATL:

- A *ProfileElement* can have more than one UML concept. We have to choose the good one when we transform a metamodel element to an UML one. In the profile for MOF, for example, UML Package and UML Model are both mapped to MOF Package. However, in the inverse direction, a MOF Package must often be mapped to UML Package only.
- A weaving link can have several left elements and one right one. In this transformation we have to filter those elements and select pertinent ones. The example of the UML *Class* that maps to MOF *Class* on the profile for MOF is a characteristic of this. The attribute *contents* of the MOF Class maps to *ownedElement* followed by feature (in order) and *taggedValue* of the UML Class. The *Nested* link which expresses this mapping has three left elements and one right one. For *T2AMWtoATL*, we have to select appropriate elements from the attribute contents of the MOF *Class*, which have to be affected to each attribute from *ownedElement*, *feature*, and *taggedValue* of the UML Class.
- If the right metamodel of the weaving does not have a root element, the ATL result model of the transformation will not have a rule to create a global UML Model. Every UML model must have a Model element, which contains all the Packages; it is like a root of the UML model. However, we need a metamodel element that can be linked with this UML element. When the metamodel does not have one, we currently have to add it manually in the resulting UML model.

5. Conclusions

DSM can be viewed as a bottom-up and UML-Profiling as a top-down approach. This has not the same meaning as in structured programming. Here we mean that in the first approach we are using small and well-focused blocks (metamodels) to capture the various needs of developers, maintainers, end-users, etc. On the contrary in the second approach, one universal language (UML) is supposed to capture most of the needs of these actors. As a consequence the first approach has to address a challenging composition and coordination problem while the second one has to address a very difficult restriction and extension problem. What is really needed for a given task is a superset of a subset of UML. The notion of a UML profile has been introduced to allow this restriction/extension operation. On the contrary, DSM approaches have still to figure out how to deal with important numbers of models expressed in different metamodels.

There are probably some possible compromises between these two radical approaches. Today the UML profiling technique is more widespread because it is supported by the main UML CASE tools vendors (like IBM/RSA or Gentleware/Poseidon), generating a lot of UML profile legacy. Specific tool-variants of profiles, various versions of standard profiles themselves based on various versions of UML (1.3, 1.4, 2.0, etc.) make the situation rather heterogeneous but in principle any UML tool should be able to work on any kind of UML profile. In the DSM camp the situation has not yet reached this level because the last generation of MetaCase tools (i.e. tools adaptable to any metamodel) are not yet widely available but this may change very rapidly.

While the debate between pure DSM and UML profiling is going on, the production of models based on both approaches is spreading. We have taken the point of view in this paper that there is a growing need for establishing bridges between both approaches. Starting from there we have looked at the possibility of using model-engineering tools to build these bridges.. Considering this problem to be a particularly real-life typical problem we have shown how model-transformation languages like ATL may help to provide a solution. One interesting conclusion is that the possibility to build and use higher order transformation (HOTs) in ATL is of high relevance. By HOTs we mean transformations taking other transformations as input and/or producing other transformations as output. The intuition that HOTs were of high conceptual importance has met practicality need in this project. Also of interest, we have found that model transformation by itself is not sufficient and should be complemented by another functionality called here model weaving. Using a corresponding tool (AMW), we have provided a complete initial prototype showing how to create a practical bridge. So the second result of this paper has been to show how the complementary techniques of model transformation and model weaving may play together to solve significant problems.

Throughout this paper we have voluntarily refrained from giving any personal opinion on the pertinence of the UML profiling technology. This is a separate debate. OMG is currently building on this UML profiling technology but could also switch at some time in the future on a more DSM-oriented path without changing its overall MDA philosophy [5]. What we can say for the time being is that DSM is still lacking tools and that UML profiling technology is very complex partly in reason of its loose initial definition. The work presented in this paper illustrates this complexity. Whatever the evolution of MDE mainstream, in one direction or the other one, we'll need practical bridging tools to reconcile both worlds or to provide migration paths. We have found this problem to be a hard one and we have used it as a significant example to illustrate the need for HOTs and practical Model Weaving support.

6. Acknowledgements

We would like to thank Freddy Allilaire, David Touzet, Patrick Valduriez and all the members of the ATLAS group for their numerous contributions to this document. This work has been supported in part by the IST European project "ModelWare" (contract 511731) and by a grant from Microsoft Research (Cambridge).

7. References

[1] Van Deursen, A, and Klint, P, Visser, J: *Domain-Specific languages: An Annotated Bibliography.* ACM SIGPLAN Notices 35(6):26-36 (2000), http://homepages.cwi.nl/~arie/papers/dslbib.pdf

[2] Cabot, J, and Gómez, C: *A simple yet useful approach to implementing UML Profiles in current case tools.* In: Workshop in Software Model Engineering, (2003), http://www.metamodel.com/wisme-2003/05.pdf

[3] Bézivin, J, Jouault, F, and Touzet, D: *Principles, Standards and Tools for Model Engineering*. In: Proceedings of the Using metamodels to support MDD Workshop, 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2005). See also research report http://www.sciences.univ-nantes.fr/lina/atl/www/papers/RR-LINA2005-01.pdf

[4] Wada, H, and Suzuki, J: *Modeling Turnpike: a Model-Driven Framework for Domain-Specific Software Development*. In: Doctoral Symposium, ACM/8th IEEE International Conference on Model Driven Engineering Languages and systems (2005), http://www.cs.colostate.edu/models05/doctoralSymposiumPapers/hiroshi.pdf

[5] OMG, *OMG Model-Driven Architecture*, OMG Document (2000), http://www.omg.org/mda

[6] Eclipse Modeling Framework http://www.eclipse.org/emf/

[7] AMMA Platform, *ATLAS Transformation Language* http://www.sciences.univ-nantes.fr/lina/atl/AMMAROOT/

[8] OMG, *UML Superstructure Specification*, version 2.0, OMG Adopted Specification ptc/03-08-02 (2003), http://www.omg.org/docs/ptc/03-08-02.pdf

[9] OMG, *UML Profile for MOF*, OMG Formal Specification formal/04-02-06 (2004), http://www.omg.org/docs/formal/04-02-06.pdf

[10] OMG, *Meta Object Facility (MOF) Specification*, version 1.4, OMG Formal Specification formal/2002-04-03 (2002), http://www.omg.org/technology/documents/formal/mof.htm

[11] OMG, *UML Profile for CORBA Components*, OMG Formal Specification formal/05-07-06 (2005), http://www.omg.org/docs/formal/05-07-06.pdf

[12] OMG, *Metamodel and UML Profile for Java and EJB Specification*, OMG Formal Specification formal/04-02-02 (2004), http://www.omg.org/cgi-bin/doc?formal/2004-02-02

[13] OMG, *UML OCL 2.0 Specification*, OMG Adopted Specification ptc/03-10-14 (2003), http://www.omg.org/docs/ptc/03-10-14.pdf

[14] Greenfield, J. Short, K., Cook, S., Kent, S, (Foreword by Crupi, J.), *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. ISBN: 0-471-20284-3 Paperback 696 pages, September 2004.