Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets, © JOT 2011
Online at http://www.jot.fm.

EMF Profiles: A Lightweight Extension Approach for EMF Models

Philip Langer[a] Konrad Wieland[a] Manuel Wimmer[b]
Jordi Cabot[c]

a. Vienna University of Technology, Austria
b. Universidad de Málaga, Spain
c. INRIA & École des Mines de Nantes, France

Abstract Domain-Specific Modeling Languages (DSMLs) are getting more and more attention as a key element of Model Driven Engineering. As any other software artifact, DSMLs should continuously evolve to adapt to the changing needs of the domain they represent. Unfortunately, right now evolution of DSMLs is a costly process that requires changing the DSML metamodel and re-creating the complete modeling environment.

In this paper we advocate for the use of EMF Profiles, an adaptation of the UML Profile concept to DSMLs. Profiles have been a key enabler for the success of UML by providing a lightweight language-inherent extension mechanism which is expressive enough to cover an important subset of extension scenarios. We believe a similar concept for DSMLs would provide a valuable extension mechanism which has been so far neglected by current metamodeling tools. Apart from direct metamodel profiles, we also propose reusable profile definition mechanisms whereby profiles are defined independently of any DSML and, later on, coupled with all DSMLs that can benefit from these profiles. Our approach has been implemented in a prototype integrated in the EMF environment.

Keywords language extensions, UML Profiles, language engineering.

1 Introduction

Domain-Specific Modeling Languages (DSMLs) have gained much attention in the last decade [KT08]. They considerably helped to raise the level of abstraction in software development by providing designers with modeling languages tailored to their application domain. However, as any other software artifact, DSMLs are continuously subjected to evolution in order to be adapted to the changing needs of the domain they represent. Currently, evolving DSMLs is a time-consuming and tedious task because not only its abstract and concrete syntax but also all related artifacts as well

Philip Langer, Konrad Wieland, Manuel Wimmer, Jordi Cabot. *EMF Profiles: A Lightweight Extension Approach for EMF Models*. In Journal of Object Technology, vol. 11, no. 1, 2012, pages 8:1–29.
doi:10.5381/jot.2012.11.1.a8

as all DSML-specific components of the modeling environment have to be re-created or adapted each and every time.

UML has avoided these problems by promoting the use of profiles. Indeed, the profile mechanism has been a key enabler for the success and widespread use of UML by providing a lightweight, language-inherent extension mechanism [Sel07]. Many UML tools allow the specification and usage of user-defined profiles and are often shipped with various pre-defined UML Profiles. Induced by their widespread adoption, several UML Profiles have even been standardized by the OMG[1].

In the last decade, many debates[2] on pros and cons of creating new modeling languages either by defining metamodels from scratch (with the additional burden of creating a specific modeling environment and handling their evolution) or by extending the UML metamodel with UML Profiles (which provide only a limited language adaptation mechanism) have been going on.

However, in this paper we propose a different solution to combine the best of both breeds. We advocate for adapting the UML Profiles concept as an extension mechanism for *existing* DSMLs. We believe the usage of profiles in the realm of DSMLs brings the following benefits.

Lightweight language extension. One of the major advantages of UML Profiles is the ability to systematically introduce further language elements without having to re-create the whole modeling environment such as editors, transformations, and model APIs. This also facilitates to test new modeling languages (or new language constructs) easily. Instead of creating a full new modeling language and a dedicated modeling environment from the beginning on, one can start with a UML Profile to validate whether users find the modeling language useful.

Dynamic model extension. In contrast to direct metamodel extensions, also already existing models may be dynamically extended by additional profile information without recreating the extended model elements. One model element may be further annotated with several stereotypes (even contained in different profiles) at the same time which is equivalent to the model element having multiple types [AK07]. Furthermore, the additional information introduced by the profile application is kept separated from the model and, therefore, does not pollute the actual model instances.

Preventing metamodel pollution. Information not coming from the modeling domain, can be represented by additional profiles without polluting the actual domain metamodels. Consider for instance annotating the results of a model review (as known from code reviewing) which shall be attached to the reviewed domain models. Metaclasses concerning model reviews do not particularly relate to the domain and, therefore, should not be introduced in the domain metamodels. Using specific profiles instead helps to separate such concerns from the domain metamodel and keeps the metamodel concise and consequently, the language complexity small.

Model-based representation. Additional information, introduced to the models by profile applications, is accessible and processable like ordinary model information. Consequently, model engineers may reuse familiar model engineering technologies to

[1] `http://www.omg.org/technology/documents/profile_catalog.htm`

[2] Consider for instance the panel discussion "A DSL or UML Profile. Which would you use?" at MoDELS'05 (`http://www.cs.colostate.edu/models05/panels.html`)

process profile applications. Due to their model-based representation, profile applications may also be validated against the profile definition to ensure their consistency as it is known from metamodel/model conformance.

Until now, and despite the previous benefits, the notion of profiles has not been adopted in current metamodeling tools. Thus, the main contribution of this paper is to adapt the notion of UML Profiles to arbitrary modeling languages, in particular those residing in the Eclipse Modeling Framework[3] (EMF) [SBPM08] which is currently one of the most popular metamodeling frameworks. Nevertheless, the ideas presented herein can be reused to add profiles support in other metamodeling platforms. Thereby, existing modeling languages may easily be extended by profiles in a similar way as it is known from UML tools. Additionally, we propose two novel techniques to enable the systematic reuse of profile definitions across different modeling languages. First, we introduce *generic profiles* which are created independently of the modeling language in the first place and may be bound later to *several modeling languages*. Second, we propose *meta profiles* for immediately reusing them for *all modeling languages*. Finally, we present how our prototype called EMF Profiles is integrated in EMF.

This article is an extension of a paper presented at the TOOLS 2011 conference [LWWC11]. Besides incremental advancements of the EMF Profile language, this article introduces two major extensions over the previous paper. First, a new section (cf. Section 3) has been added, which highlights the design decisions taken for EMF Profile and provides a detailed comparison between UML Profiles and EMF Profiles. Furthermore, Section 5 has been extended significantly. In particular, we introduce the EMF Profiles API and show how EMF Profiles can be integrated easily with other EMF technologies, such as the model transformation language ATL.

2 From UML Profiles to EMF Profiles

In this section, we present the *standard profile* mechanism (as known from UML) for EMF. Therefore, we disclose our design principles and we discuss how the profile mechanism may be integrated in EMF in a way that profiles can seamlessly be used within EMF following our design principles. Finally, we show how profiles as well as their applications are represented based on an example.

2.1 Design Principles

With EMF Profiles we aim at realizing the following five design principles. Firstly, annotating a model should be as *lightweight* as possible, hence, no adaptation of existing metamodels should be required. Secondly, we aim at avoiding to *pollute* existing metamodels with concerns not directly related to the modeling domain. Thirdly, we aim at *separating annotations from the base model* to allow importing only those annotations which are of current interest for a particular modeler in a particular situation. Fourthly, the annotations shall be *conforming to a formal and well-known specification* such as it is known from metamodel/model conformance. Finally, users should be enabled to *intuitively attach annotations* using environments and editors they are familiar with. Consequently, annotations shall be created either on top of the concrete (graphical) syntax of a model or on top of the abstract syntax using e.g., generic tree-based editors.

[3] `http://www.eclipse.org/modeling/emf`

2.2 Integrating Profiles in the EMF Metalevel Architecture

The profile concept is foreseen as an integral part of the UML specification. Therefore, the UML package *Profiles*, which constitutes the language for specifying UML Profiles, resides, in terms of the metamodeling stack [Küh06], at the meta-metalevel M_3 [OMGc] as depicted in Fig. 1. A specific profile (*aProfile*), as an instance of the meta-metapackage *Profile*, is located at the metalevel M_2 and, therefore, resides on the same level as the UML metamodel itself. Thus, modelers may create profile applications (*aProfileApplication* on M_1) by instantiating *aProfile* just like any other concept in the UML metamodel.

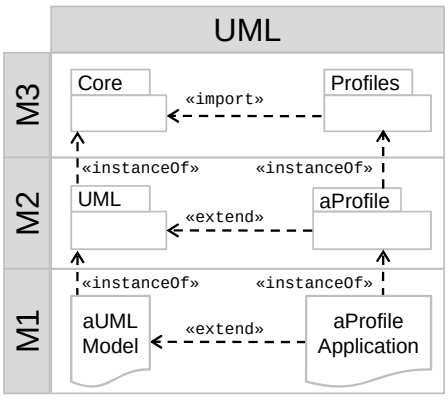

Figure 1 – UML Architecture

To embed the profile mechanism into EMF, a language (equivalent to the package *Profiles* in Fig. 1) for specifying profiles is needed as a first ingredient. This is easily achieved by creating an Ecore-based metamodel which is referred to as *Profile MM* (cf. column *Profile Definition* in Fig. 2). Specific profiles, containing stereotypes and tagged values, may now be modeled by creating instances, referred to as *aProfile*, of this profile metamodel. Once a specific profile is at hand, users should now be enabled to apply this profile to arbitrary models by creating *stereotype applications* containing concrete values for tagged values defined in the stereotypes. As already mentioned, in UML, a stereotype application is an instance—residing on M_1—of a stereotype specification in M_2 (cf. Fig. 1).

Unfortunately, in contrast to the UML architecture, in EMF no profile support exists in M_3. The level M_3 in EMF is constituted only by the metamodeling language Ecore (an implementation of MOF [OMGa]) which has no foreseen profile support. Extending Ecore on level M_3 to achieve the same instantiation capabilities for profiles as in UML is not a desirable option, because this would demand for an extensive intervention with the current implementation of the standard EMF framework. Therefore, in EMF, our profile metamodel (*ProfileMM* in column *Profile Definition* of Fig. 2) is defined at level M_2 and the user-defined profiles (*aProfile*) reside on M_1. As an unfortunate result, a defined stereotype in *aProfile* cannot be instantiated for representing stereotype applications (as in UML), because *aProfile* is already located on M_1 and EMF does not allow for *instantiating an instance* of a metamodel, i.e., EMF does not directly support multilevel modeling [AK01].

Therefore, more sophisticated techniques have to be found for representing stereotype applications in EMF. In particular, we identified two strategies for lifting *aProfile* from M_1 to M_2 in order to make it instantiable and directly applicable to EMF models.

1. **Metalevel Lifting By Transformation.** The first strategy is to apply a model-to-model transformation which generates a metamodel on M_2, corresponding to the specified profile on M_1. The generated metamodel, denoted as *aProfile as MM* in column (*a*) of Fig. 2, is established by implementing a mapping from Profile concepts to Ecore concepts. In particular, the transformation generates for each `Stereotype` a corresponding `EClass` and for each `TaggedValue` a corresponding `EStructuralFeature`. The resulting metamodel is a direct instance of Ecore residing on M_2 and therefore, it can be instantiated to represent profile applications.

2. **Metalevel Lifting By Inheritance.** The second strategy allows to *directly* instantiate profiles by *inheriting instantiation capabilities* (cf. «*inheritsFrom*» in column (*b*) of Fig. 2). In EMF, only instances of the meta-metaclass `EClass` residing on M_3 (e.g., the metaclass `Stereotype`) are instantiable to obtain an object on M_1 (e.g., a specific stereotype). Consequently, to allow for the direct instantiation of a defined stereotype on M_1, we specified the metaclass `Stereotype` in *Profile MM* to be a *subclass of* the meta-metaclass `EClass`. By this, a stereotype inherits EMF's capability to be instantiated and thus, a stereotype application may be represented by a direct instance of a specific stereotype.

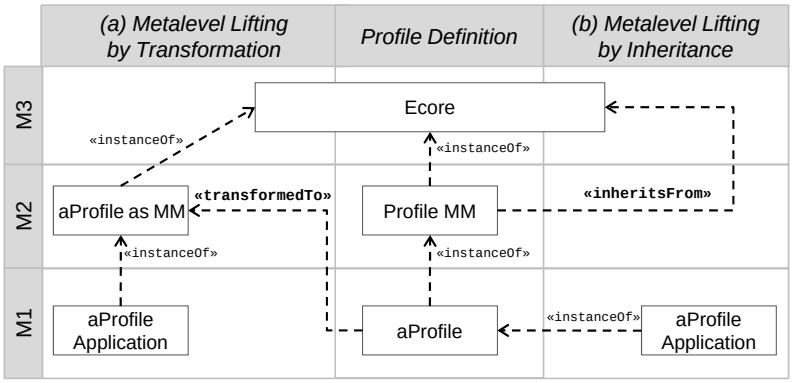

Figure 2 – EMF Profile Architecture Strategies

We decided to apply the second strategy, because of the advantage of using only one artifact for both, (1) defining the profile and (2) for its instantiation. This is possible because by this strategy, a profile is now a dual-faceted entity regarding the metalevels which is especially obvious when considering the horizontal «*instanceOf*» relationship between *aProfile* and *aProfileApplication* (cf. Fig. 2). On the one hand, a profile is located on M_1 when considering it as an instance of the profile metamodel (*ProfileMM* on M_2)). On the other hand, the stereotypes contained in the profile are indirect instances of `EClass` and are therefore instantiable which means that a profile may also be situated on M_2. Especially, when taking the latter view-point, the horizontal «*instanceOf*» relationship between *aProfile* and *aProfileApplication* shown in Fig. 2 will become the expected vertical relationship as in the UML metalevel architecture.

2.3 The EMF Profile Metamodel

The metamodel of our profile definition language is illustrated in package *Standard EMF Profile* depicted in Fig. 3. As a positive side effect of choosing the metalevel lifting strategy 2, the class `Stereotype`, being a specialization of `EClass`, may also contain `EAttributes` and `EReferences`, which are reused to represent tagged values. Thus, no dedicated metaclasses have to be introduced to represent the concept of tagged values. To specify the applicability of stereotypes to metaclasses, the class `Stereotype` comprises a reference to the class `Extension`. Thereby, users may define the base metaclass of the stereotype, as well as the lower and upper bound of a stereotype application. For instance, a lower bound of `1` in an `Extension` indicates that the respective stereotype must be applied to each instance of the base metaclass in order to obtain a valid profile application. Besides the upper and lower bound, users may *redefine* or *subset* extension relationships of superstereotypes by setting the reference `redefined` or `subsetted`, respectively. With these two references, we adopt the modeling features known from `Associations` in UML (cf. Section 3.1 for more details).

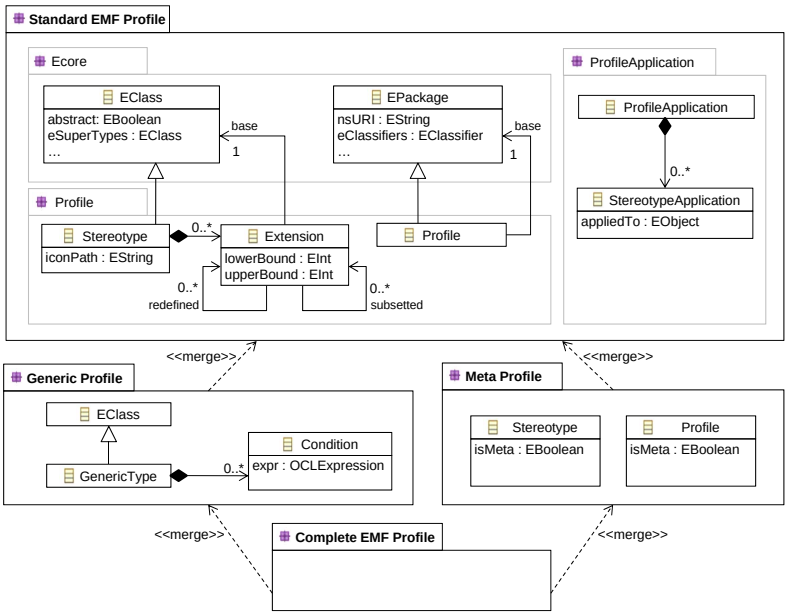

Figure 3 – EMF Profile Metamodel

Stereotype applications require to have a reference to the model elements to which they are applied. Therefore, we introduced an additional metamodel package, namely `ProfileApplication` in Fig. 3. This metamodel package contains a class `StereotypeApplication` with a reference to arbitrary `EObjects` named `appliedTo`. Whenever, a profile is saved, we automatically add `StereotypeApplication` as a superclass to each specified stereotype. To recall, this is possible because each `Stereotype` is an instance of `EClass`, which may have superclasses. Being a subclass of `StereotypeApplication`, stereotypes inherit the reference `appliedTo` automatically. In the following subsection, we further elaborate on the EMF Profile metamodel by providing a concrete example. Please note that the so far unmentioned packages *Generic Profile* and *Meta Profile* in Fig. 3 are discussed in Section 4. Furthermore, for presentation purposes, we have used the package merge [DDZ08] from UML to structure the language features of EMF

Profiles into different packages. The UML package merge allows to incrementally define modeling languages. In our example, all language features are finally merged into the *Complete EMF Profile* package representing the complete EMF Profile language.

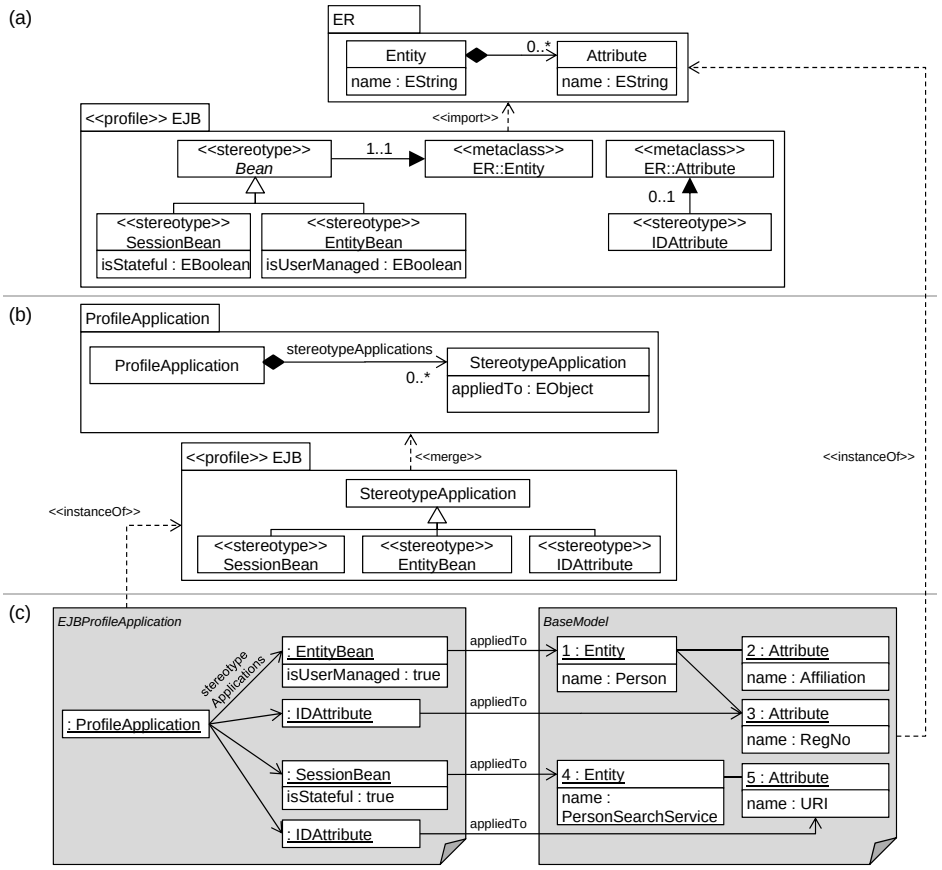

Figure 4 – EMF Profiles by Example: (a) Profile Definition User-View, (b) Internal Profile Representation, (c) Profile Application

2.4 Applying the EMF Profile Metamodel

To clarify how profiles and profile applications are represented from a technical point of view, we make use of a small example. In particular, a simplified version of the well-known *EJB profile* is applied to an Entity-Relationship (ER) model [Che76]. Fig. 4(a) depicts an excerpt of the ER metamodel and the EJB profile. The EJB profile contains the abstract stereotype `Bean`, which extends the metaclass `Entity` of the ER metamodel, and two concrete substereotypes of `Bean` called `SessionBean` and `EntityBean`. Besides, the profile introduces the stereotype `IDAttribute` extending the metaclass `Attribute` to indicate the ID of an `Entity`.

As already mentioned in the previous subsection, internally, we use the *ProfileApplication* metamodel (cf. Fig. 4(b)) to weave the necessary concepts for a profile's application into a profile model. In particular, the class `ProfileApplication` acts as root element for all `StereotypeApplications` in a profile application model. Furthermore, all `Stereotypes` inherit the reference `appliedTo` from `StereotypeApplication`. When instantiating (i.e., applying) the EJB profile, a root element of the type `Profile-`

`Application` is created which may contain stereotype applications as depicted in Fig. 4(c). For determining the applicability of a stereotype `s` to a particular model element `m`, it is checked whether the model element's metaclass (`m.eClass()`) is included in the list of metaclasses that are extended by the stereotype (`s.getBase()`). If so, the stereotype `s` is applicable to model element `m`. Each stereotype application is represented as a direct instance of the respective stereotype (e.g., «*EntityBean*») and refers to the model element in the *BaseModel* to which it is applied by the reference `appliedTo` (inherited from the class `StereotypeApplication`). Please note that the EJB profile application resides in a separated model file and not in the original ER model denoted with *BaseModel* in Fig. 4.

3 Comparing UML Profiles with EMF Profiles

With the introduction of UML 2, the profile mechanism has been tremendously improved compared to the beginnings of UML. In UML 1.1, stereotypes and tagged values were represented by String-based values, which could be attached to UML model elements. With the evolution of UML, specific modeling support has been included in the UML standard to provide precise definitions of UML Profiles. In particular, a Profile modeling language has been incorporated in the UML language family. In this section, we first revisit the modeling concepts of the UML Profile modeling language by summarizing the key definitions of the UML standard. Next, we report on experiences gained in evaluating four major UML modeling tools with respect to the implementation of UML Profiles, and finally, we highlight the differences between EMF Profiles and UML Profiles according to the UML standard.

3.1 UML Profile Modeling Language Revisited

The `Profile` package of the UML metamodel (cf. Fig. 5 for an excerpt) defines the UML Profile modeling language and is based on the packages defining the UML class diagram language (cf. `Classes` package in Fig. 5). In particular, the `Profile` package introduces four additional classes for defining the modeling concepts of UML Profiles. The classes `Profile` and `Stereotype` are derived from the classes `Package` and `Class`, respectively. For representing the extension relationships between metaclasses and stereotypes, the class `Extension` is defined as a subclass of the class `Association` with the constraint that instances of the `Extension` class are only allowed to be binary relationships, i.e., they must have exactly two association ends. An association end in UML is represented by the class `Property`. One of the association ends has to be of type `ExtensionEnd` which is a subclass of the class `Property` and which has to be linked to stereotypes, only. Extensions may be defined to be required. Thereby, each instance of the referred metaclass needs a stereotype application assigned to form a valid model.

To elaborate the representation of extension relationships in UML in more detail, Fig. 6 introduces an example which is based on the UML standard (cf. page 183 in [OMGc]). In this figure, the extension relationship between the metaclass `C1` and the stereotype `S1` is illustrated in the *extension notation*, as well as in the *association notation*, whereas both notations describe the same extension relationship. The association notation is applicable, because an extension relationship is an association in the UML metamodel (cf. Fig. 5). In particular, the extension relationship is represented by an association with two association ends. The association end `base_C1`

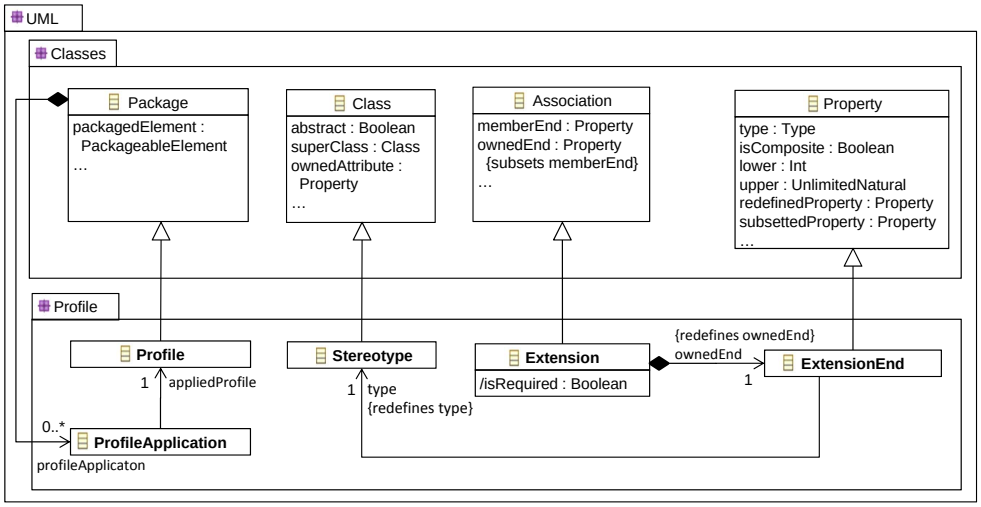

Figure 5 – Excerpt of the UML Profile Package

must have a multiplicity of `1`, which means that a stereotype application always must be linked to exactly one base element. The opposite association end is defined to be composite. Consequently, the stereotype application is automatically deleted when the base element is deleted. The extension relationship in this example is not defined as required, as the multiplicity is `0..1`. However, UML also foresees required extension relationships using a multiplicity of `1..1`.

For the introduced modeling concepts of the UML Profile package, several OCL constraints are defined which restrict the possibilities of the UML Profile modeling language with respect to UML class diagrams. In particular, the following three OCL constraints are heavily influencing how stereotypes are *defined* and *applied* in profile applications:

Constraint 1. The multiplicity of `ExtensionEnds` in UML are `0..1` or `1` (cf. Constraint [1] on page 181 in [OMGc]). This constraint implies that a stereotype is only applicable once to the same base element.

Constraint 2. The aggregation of `ExtensionEnd` is always composite (cf. Constraint [2] on page 181 in [OMGc]). This constraint implies that a stereotype application is deleted when the base element is deleted.

Constraint 3. A stereotype may only generalize or specialize another stereotype (cf. Constraint [1] on page 198 in [OMGc]). This constraint implies that a stereotype

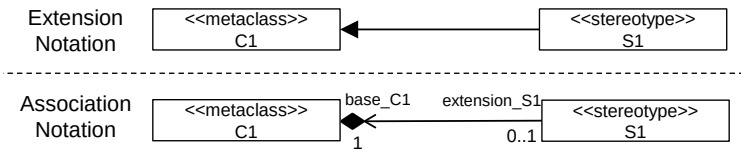

Figure 6 – Example of an Extension Relationship

is not allowed to inherit from common metaclasses.

Constraint 4. Stereotype is the only kind of metaclass that cannot be extended by stereotypes (cf. page 180 in [OMGc]). Thus, it is not possible to apply stereotypes to existing stereotype applications. Profile applications are considered to be flat models only covering one level which subsumes all applied stereotypes.

The afore presented modeling concepts together with the mentioned constraints are implemented by most of the currently available UML tools. We have surveyed four of the major players in this field, namely Magic Draw[4], UML2 Eclipse project[5], EnterpriseArchitect[6], and Papyrus[7]. However, while testing these UML modeling tools regarding the specification and application possibilities for UML Profiles several issues have been identified that are enumerated and exemplified in the following. It has to be emphasized that these issues are not explicitly treated by the current UML standard.

Issue 1. Stereotypes cannot contain complex data in form of new objects which are directly possessed by a profile application. Stereotypes can only refer to elements in the base model and to other stereotypes, but cannot act as containers for specifying their own internal object structure. The UML standard does not explicitly treat this issue. Only one sentence in the standard can be found (cf. page 190 in [OMGc]) giving a hint that a profile may actually contain classes. However it is not mentioned if the instances of these classes should be contained by the base model or by the profile application. Therefore, the example shown in Fig. 7 is ambiguous. The surveyed modeling tools do not allow to define instances of the class `C2` in the profile application. The only way to instantiate this class is to create objects in the base model.

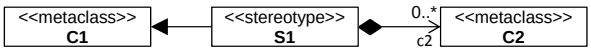

Figure 7 – Issue 1: Stereotype as Container

Issue 2. The association semantics, more specifically the multiplicities, of extension relationships discussed before are not always fulfilled by current UML tools when inheritance hierarchies among stereotypes are defined. For example, consider the profile definition in Fig. 8(a). In current UML tools, the profile application model depicted in Fig. 8(c) is a valid instance of the defined UML Profile shown in Fig. 8(a). However, when we consider the equivalent association representation of the extension relationship illustrated in Fig. 8(b), we see that instances of class `C1` are only allowed to have one link to stereotypes of kind of `S1`.

Issue 3. Extension relationships are defined as associations in the UML standard. Thus, advanced modeling features for associations, such as refinements of properties (cf. features `redefinedProperty` and `subsettedProperty` in the UML metamodel shown in Fig. 5), may also be used in profile specifications. However, the surveyed UML modeling tools disallow or ignore these features when applying the specified profiles to the base models. For example, Fig. 9 shows two extension relationships, whereas the extension relationship between `C2` and `S2` is a refinement of the extension

[4] `https://www.magicdraw.com`
[5] `http://www.eclipse.org/uml2`
[6] `http://www.sparxsystems.com`
[7] `http://www.eclipse.org/papyrus`

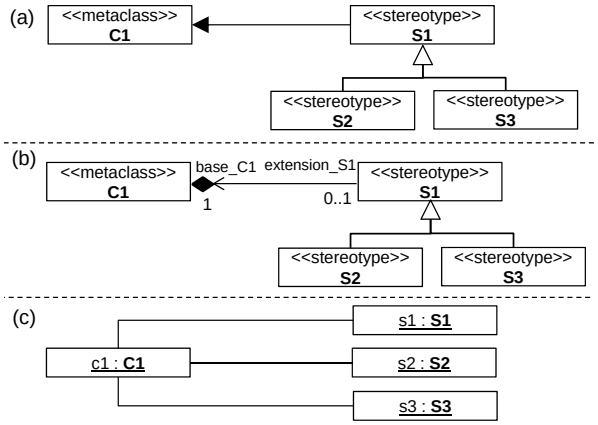

Figure 8 – Issue 2: Extension Relationship of Superstereotypes implicitly duplicated for Substereotypes: (a) Profile Example, (b) Profile Example shown in Association Notation, (c) Profile Application (Valid in current UML Tools, Invalid w.r.t (b))

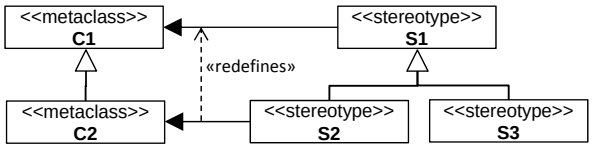

Figure 9 – Issue 3: Advanced Features of Extension Relationships

relationship between `C1` and `S1`. Thus, only the stereotype `S2` should be applicable to instances of the class `C2`. In the surveyed UML tools, however, instances of class `C2` may be annotated by all kinds of `S1` stereotypes in addition to the stereotype `S2`.

3.2 Commonalities and Differences between UML Profiles and EMF Profiles

In the following, we compare the EMF Profiles implementation with the UML standard, as well as with existing implementations of profiles in UML modeling tools. In particular, we focus on highlighting differences due to the retroactive integration with EMF as well as intended differences of EMF Profiles and UML Profiles based on the previously discussed constraints and issues.

The basic structure of EMF Profiles and UML Profiles is very similar. As in UML, the modeling concepts in EMF Profiles for specifying profiles are also contained in a package called `Profile` (cf. Fig. 3) and largely correspond to the modeling concepts in the package `Profile` in UML. Moreover, as in UML, the class `Profile` in EMF Profiles is a subclass of the class `EPackage`, the class `Stereotype` is a subclass of `EClass`, and tagged values are modeled in terms of properties of an `EClass`.

However, there are also differences between EMF Profiles and UML Profiles. Some of these differences stem from the fact that the profile support has been integrated into EMF retroactively, others are intended differences in order to enhance the expressiveness of the profile modeling language.

3.2.1 Differences due to retroactive integration in EMF

As profiles are tightly integrated into UML and foreseen in UML editors, stereotype applications may directly be included in the model to which a profile is applied. In EMF Profiles, stereotype applications have to be stored in a separate model, because EMF-based editors are not capable of handling stereotype applications in a model natively. Therefore, the class `ProfileApplication` acts as a root container for stereotype applications in a separate model. The class `ProfileApplication` in the UML metamodel, however, plays a different role; it is only used for indicating that a profile is applied to a specific package and does not contain applied stereotypes.

Another difference concerns the representation of the extension relationship. In UML, an extension's association end at the side of the metaclass is defined to be composite. As a result, the deletion of the extended model element causes the deletion of the stereotype application (cf. Constraint 2 in Section 3.1). This, however, is not possible in EMF Profiles, because containment references (being EMF's equivalent to composition in UML) are owned by the metaclass. As we may not add such additional containment references to an existing metaclass that should act as base class for a stereotype, we may only refer from the stereotype application to the extended model element (and not vice versa). Thus, if an extended model element is deleted, the stereotype application still exists without being applied to any model element. To address this issue, we provide a clean-up function, which deletes stereotype applications, if their base elements has been deleted. Please note that this might also be an advantage in certain scenarios, because stereotype applications may be detached from their base model elements temporarily and attached to another base model element again at a later time. This is not foreseen in UML.

3.2.2 Intended differences

Besides the differences stemming from the retroactive integration of profiles into EMF, there are also *intended differences* between UML Profiles and EMF Profiles. In the following, we discuss these differences with respect to the constraints of UML Profiles in the UML standard and issues in the UML modeling tools discussed in Section 3.1.

Multiple stereotype applications on one model element. According to Constraint 1 in Section 3.1, *one* stereotype may only be applied *once* to the same model element, because of the dictated multiplicity of `0..1` or `1..1` of `ExtensionEnds`. This is a reasonable design decision as long as profiles are only used as an additional *classification mechanism* [AKHS03] for existing model elements. However, if stereotypes are used as *annotation mechanism* [FGT+10], it might thoroughly be necessary to assign the same stereotype more often. An example for such a requirement is a *model review profile* for documenting the result of the examination of a model (cf. Section 4.2 for a concrete example). To enable a multiple application of one stereotype to the same model element, we decided to allow the same possibilities for lower and upper bounds in extension relationships of stereotypes as for associations. Thereby, a higher expressiveness of the profile specification language is achieved, as users may assign multiplicities different to `0..1` and `1..1`.

Stereotype may inherit from classes. As discussed in the Constraint 3 in Section 3.1, UML does not allow stereotypes to inherit from common metaclasses. In EMF Profiles, we refrain from adopting this restriction. Thus, users may specify

stereotypes that are a subclass of common metaclasses (i.e., `EClasses`) in order to let the stereotype inherit the modeling features of existing or newly created metaclasses.

Containments in stereotype applications. The UML standard does not make an explicit statement whether stereotypes are allowed to contain complex data in the form of new objects. The surveyed UML modeling tools, however, do not enable users to specify metaclasses being contained by stereotypes (cf. Issue 1 in Section 3.1). Nevertheless, we believe that this is a valuable feature because it facilitates to introduce also more complex data structures to existing models than only simple tagged values. Thus, EMF Profiles allows to create containment references in stereotypes. However, please note that EMF Profiles adopts the restriction of UML that stereotypes cannot be applied again to stereotype applications (cf. Constraint 4 in Section 3.1).

Exact interpretation of association semantics. In contrast to existing UML modeling tools, we rigorously implemented the association semantics in EMF Profiles (cf. Issue 2 in Section 3.1). For instance, consider the example shown in Fig. 8(a): if an upper bound of `1` is assigned to an extension of a stereotype, the stereotype *or one of its substereotypes* may only be applied *once* to the same model element; it is disallowed to apply the stereotype *and* one of its substereotypes at the same time to the same model element. If this would be intended, users may *redefine* the extension in a substereotype as discussed in the following or increase the upper bound.

Advanced modeling features for extensions. Another aspect of the association semantics that is not properly covered by current UML modeling tools is the possibility to redefine inherited extensions of superstereotypes in substereotypes (cf. Issue 3 in Section 3.1). Redefining the extension, however, is a valuable feature, because it allows stereotypes to inherit tagged values but specify their own specialized applicability.

Summary. In this section, we presented the differences between EMF Profiles and UML Profiles, as well as existing UML modeling tools that implement the profile mechanism. We described the differences that stem from the retroactive integration of the profile mechanism into EMF and discussed the intended deviations from the UML standard. In particular, we highlighted the aspects that are not covered in existing UML modeling tools but rigorously implemented in EMF Profiles according to the UML standard. Thereby, users are empowered to fully exploit the expressiveness of profiles. We believe that the taken design decisions lead to more precise profile definitions by using well-known concepts available in current metamodeling languages.

4 Inter-Language Profile Reuse

Originally, the profile mechanism was specifically developed for UML. Hence, profiles could only extend the UML metamodel. In the previous sections, we showed how this lightweight extension mechanism is ported to the realm of DSMLs. However, in this realm a whole pantheon of different DSMLs exists which are often concurrently employed in a single project. As a result, the need arises to reuse existing profiles and apply them to *several DSMLs*. Thus, we introduce two dedicated reuse mechanisms for two different scenarios:

1. **Metamodel-aware Profile Reuse.** The first use case scenario is when users aim to apply a profile to a *family of DSMLs* sharing similar modeling concepts.

Being aware of these modeling concepts, the user wants to take control of the applicability of stereotypes to a manually selected set of metaclasses.

An example for this scenario is a profile for annotating models defined with structural modeling languages, such as Ecore, ER, or UML class diagrams, with platform specific information. Assume we have a stereotype for marking transient classes, i.e., their objects are not stored persistently in the database. Thus, this stereotype should be applicable on instances of `Ecore::EClass`, `ER::EntityType`, and `UML::Class`, but the stereotype should be of course not be applied on other elements such as attributes or associations.

2. **Metamodel-agnostic Profile Reuse.** In the second use case scenario, users intend to use a profile for *all DSMLs* without the need for further constraining the applicability of stereotypes. Therefore, a stereotype shall—agnostic of the DSMLs' metamodels—be applicable to every existing model element.

   An example for this scenario would be a profile for marking model elements which require a revision. Thus, a *RevisionRequired* stereotype may be defined to be applicable to *every* model element, irrespectively of the type of the model element and the used modeling language.

To address scenario (1), we introduce *generic profiles* allowing to specify stereotypes that extend so-called generic types. These generic types are independent of a concrete metamodel and may be bound to specific metaclasses in order to reuse the generic profile for several metamodels. For supporting scenario (2), we propose *meta profiles* which may immediately be applied to all DSMLs implemented by an Ecore-based metamodel.

4.1 Generic Profiles

The goal behind generic profiles is to reuse a profile specification for several "user-selected" DSMLs. Therefore, a profile should not depend on a specific metamodel. Inspired by the concepts of generic programming [MS89], we use the notion of so-called *generic types* instead. In particular, stereotypes within a generic profile do not extend concrete metaclasses as presented in the previous section, they extend generic types instead. These generic types act as placeholders for concrete metaclasses in the future. Once, a user decides to use a generic profile for a specific DSML, a binding is created which connects generic types to corresponding concrete metaclasses contained in the DSML's metamodel. For one generic profile there might exist an arbitrary number of such bindings. Consequently, this allows to reuse one generic profile for several DSMLs at the same time. Furthermore, it enables users to first focus on the development of the profile and reason about the relationship to arbitrary DSMLs in a second step.

As example, consider the same EJB profile which has been specified in terms of a concrete profile in Section 2. Now, we aim at specifying the same profile in a generic way to enable its use also for other DSMLs. In particular, we show how the EJB profile may first be specified generically and we subsequently illustrate the binding of this generic profile again for ER models. We get the same modeling expressiveness as before but now in a way that allows us to reuse the EJB profile when using other data modeling languages. The original EJB profile for ER extends two metaclasses, namely the stereotypes `SessionBean` and `EntityBean` extend the metaclass `Entity` through the abstract stereotype `Bean`, and the stereotype `IDAttribute` extends `Attributes` (cf. Fig. 4). To turn this concrete profile into a generic one, we now use two generic

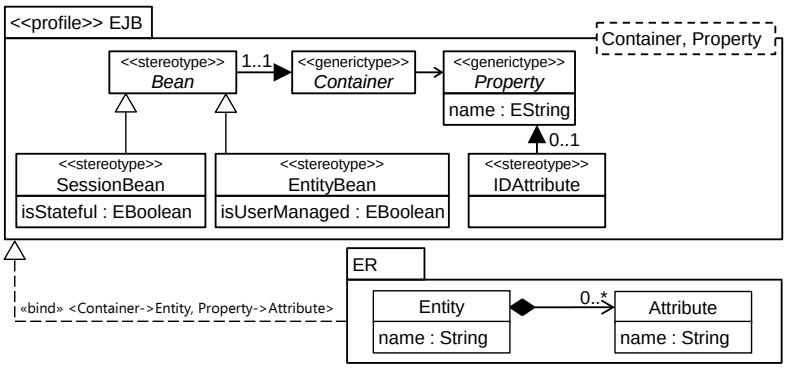

Figure 10 – Generic EJB Profile and its Binding to the ER metamodel

types, named `Container` and `Property` in Fig. 10, instead of the two concrete types `Entity` and `Attribute`.

Before we describe how generic profiles may be bound to concrete DSMLs, we first discuss conditions constraining such a binding. When developing a concrete profile, the extended DSML is known and consequently only suitable metaclasses are selected to be extended by the respective stereotypes. For instance, in the concrete EJB profile for ER, the class `Entity` may be annotated with the stereotype `EntityBean`. For marking the `Entity`'s ID attribute, the EJB profile introduces the stereotype `IDAttribute` which extends the class `Attribute`. This is reasonable, because we are aware of the fact that instances of the class `Entity` *contain* instances of the class `Attribute` in the ER metamodel, otherwise it obviously would not make any sense to extend the metaclass `Attribute` in this matter. However, generic profiles are developed without a concrete DSML in mind. Hence, profile designers possibly need to specify conditions enforcing certain characteristics to be fulfilled by the (up to this time) unknown metaclasses to which a generic type might be bound in future.

Therefore, EMF Profiles allows to attach constraints to generic profiles, which limit the metaclasses that may be bound to the respective generic types. To realize these constraints, OCL invariants are generated for each generic type, whereas the name of the generic types are masked using curly brackets in the OCL code (e.g., `{Container}`). When a user intends to bind an arbitrary metaclass to a generic type, the occurrences of the masked generic type name in the OCL constraint are replaced with the actual name of the arbitrary metaclass. If the resulting OCL constraint is fulfilled for a metaclass, the metaclass can be bound to the generic type. The bodies of the invariants can be specified either manually or semi-automatically. The semi-automatic specification of constraints is done by simply adding references or attributes to the generic types in the generic profile. When a reference or attribute is added to a generic type, a profile designer indicates implicitly that there must be a corresponding reference or attribute in the metaclass that is bound to the generic type. To realize these implicit constraints in OCL, for each reference or attribute added to the generic type, a dedicated expression is added automatically to the invariant, which specifies that the respective reference or attribute must exist. Therefore, we use the semi-automatic OCL condition generation process that has been introduced in [BLS+09]. This process allows to derive OCL conditions from model elements automatically. The derived OCL conditions reflect each feature value that has been

```
1 context {Container} inv:
2 self.eReferences->exists(r | r.eType = {Property})
3 context {Property} inv:
4 self.eAttributes->exists(a | a.name = "name" and a.eType = EString)
```

Listing 1 – OCL Constraints generated for `Container` and `Property`

set in the analyzed model elements, as well as their references to each other. As these generated conditions may not always reflect the intended conditions exactly, users may fine-tune these conditions by adding or deactivating certain parts of the generated conditions using a dedicated user interface.

In our example in Fig. 10, the profile designer created a reference in the generic type `Container`, as well as an attribute `name` in `Property`. After applying the automatic constraint generation process to this generic profile and fine-tuning the generated conditions by the user, we may obtain the OCL constraints in Listing 1.

Once the stereotypes and generic types are created, the profile is ready to be bound to concrete DSMLs. This is simply achieved by selecting suitable metaclasses of a DSML for each generic type. In our example depicted in Fig. 10, the generic types `Container` and `Property` are bound to the metaclasses `Entity` and `Attribute` in the ER metamodel, respectively, in order to allow the application of the generic EJB profile to ER models. When the binding is established, it can be persisted in two different ways. The first option is to generate a concrete profile out of the generic profile for a specific binding. This concrete profile may then be applied like a normal EMF profile as discussed in Section 2. Although this seem to be the most straightforward approach, the explicit trace between the original generic profile and the generated concrete profile is lost. Therefore, the second option is to persist the binding directly in the generic profile definition. Whenever a user intends to apply a generic profile to a concrete DSML, the EMF Profile framework searches for a persisted binding for the concrete DSML's metaclasses within the profile definition. If a binding exists, the user may start to apply the profile using this persisted binding. Otherwise, the user is requested to specify a new binding.

To support generic profiles, we extended the EMF Profile metamodel by the class `GenericType` (cf. Fig 3). Generic types inherit from `EClass` and may contain `Conditions` representing more complex constraints going beyond the aforementioned enforced references and attributes for bound metaclasses.

4.2 Meta Profiles

With meta profiles we tackle a second use case for reusing profiles for more than one DSML. Instead of supporting only a manually selected number of DSMLs, with meta profiles we aim at reusing a profile for *all* DSMLs without the need of defining an explicit extension for each DSML. This is particularly practical for profiles enabling general annotations which are suitable for every DSML. In other words, stereotypes within a meta profile must be agnostic of a specific metamodel and shall be applicable to *every model element* irrespectively of its metaclass, i.e., its type.

In EMF, every model element is an instance of a metaclass. Each metaclass is again an instance of Ecore's `EClass`. Therefore, meta-stereotypes in a meta profile do not extend metaclasses directly. Instead, they are configured to be applicable to *all instances of instances* of `EClass` and, consequently, to every model element (as

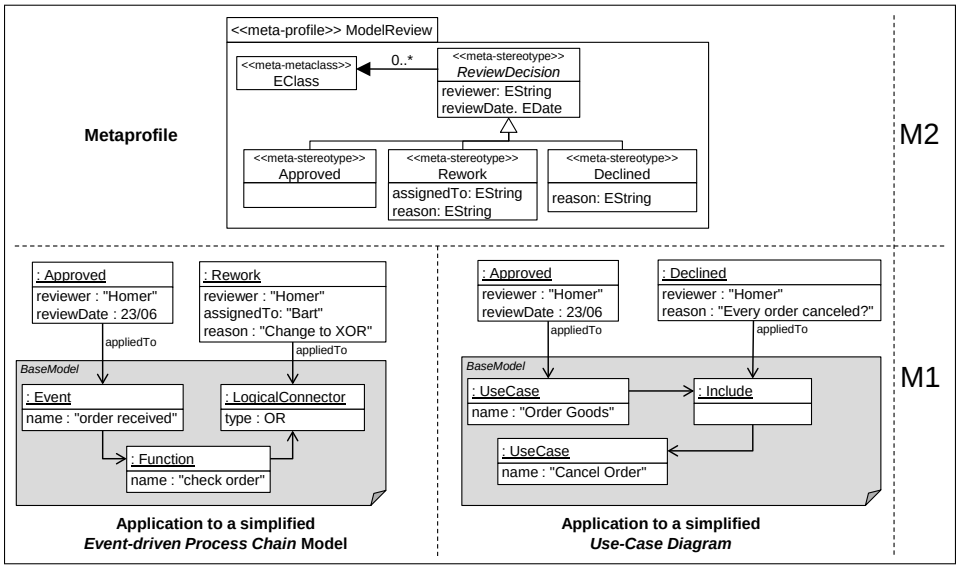

Figure 11 – Meta Profile Example: The Model Review Profile

an instance of an instance of `EClass`). This approach is inspired by the concept of potency known from multilevel metamodeling [AK01]. Using the notion of potency, one may control on which metamodeling level a model element may be instantiated. By default, the potency is 1 which indicates that a model element may be instantiated in the next lower metamodeling level. By a potency $p \geq 1$ on a metamodeling level n, a model element may be configured to be also instantiable on the level $n - p$ instead of the next lower level only. In terms of this notion of potency, a meta-stereotype has a potency of $p = 2$.

Meta profiles are created just like normal profiles. However, a new attribute, namely `isMeta`, is introduced to the profile metamodel for indicating whether a stereotype is a meta-stereotype (cf. Fig. 3). The Boolean value of this attribute is considered by EMF Profiles when evaluating the applicability of stereotypes. In particular, if `isMeta` is `true`, a stereotype is always considered to be applicable to every model element, irrespectively of its metaclass.

Our example for presenting metaprofiles is a *model review profile* (cf. Fig. 11). The goal of this profile is to allow for annotating the results of a systematic examination of a model. Since every model irrespectively of its metamodel can be subject to a review, this profile is suitable for every DSML. For simplicity, we just introduce three stereotypes in the review profile, namely `Approved`, `Rework`, and `Declined`, which shall be applicable to every kind of element in every DSML. Therefore, these three stereotypes extend the class `EClass` and are marked as meta-stereotypes (indicated by `meta-stereotype` in Fig. 11). By this, the applicability of these stereotypes is checked by comparing the meta-metatypes of model elements with the metaclasses extended by the stereotypes. As a result, the metaprofile in our example is applicable to every element in every DSML.

In the example shown in Fig. 11, we depicted the Object Diagram of two separate applications of the same metaprofile to two models conforming to different metamodels. In the first Object Diagram, an `Event` and one `LogicalConnector` within an Event-

driven Process Chain (EPC) model have been annotated with the meta-stereotype `Approved` and `Rework`, respectively. This is possible because both instances in the EPC model are an instance of a metaclass which is again an instance of `EClass`. The same metaprofile may be also applied to any other modeling language. Of course, also UML itself is supported by EMF Profiles. Therefore, the model review profile may be also applied to, for example, a UML Use Case Diagram (cf. Fig. 11). In this figure, the stereotype `Approved` has been assigned to the `UseCase` named "`Order Goods`" and the stereotype `Declined` has been applied to the `Includes` relationship.

4.3 Summary

Both techniques for enabling the reuse of profiles for several DSMLs have their advantages and disadvantages depending on the intended use case. Meta profiles are immediately applicable to all DSMLs without further user intervention. However, with meta profiles no means for restricting the use of such profiles for concrete DSMLs exist. If this is required, generic profiles are the better choice. When specifying generic profiles, explicit conditions may be used to control a profile's usage for concrete DSMLs. On the downside, this can only be done with additional efforts for specifying such conditions in the generic profile and creating manual bindings from generic profiles to concrete DSMLs.

5 A Tour on EMF Profiles

In this section, we present our prototypical implementation of EMF Profiles which is realized as Eclipse plug-in on top of the Eclipse Modeling Framework (EMF) and Graphical Modeling Framework[8] (GMF). Please note that we refrained from modifying any artifact residing in EMF or GMF. EMF Profiles only uses well-defined extension points provided by these frameworks for realizing profile support within the EMF ecosystem. For a screencast of EMF Profiles, we kindly refer to our project homepage[9].

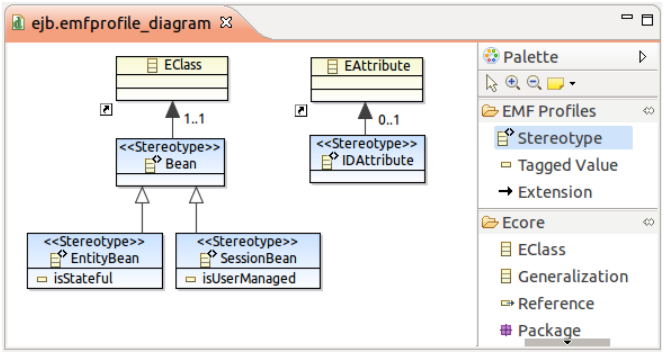

Figure 12 – EJB Profile defined with EMF Profiles Diagram Editor

Profile Definition. To define a profile, modelers may apply either the tree editor automatically generated from the Profile Metamodel or our graphical EMF Profiles Diagram Editor which is realized with GMF (cf. Fig. 12 for a screenshot). The graphical

[8] `http://www.eclipse.org/gmf`
[9] `http://www.modelversioning.org/emf-profiles`

notation used in this editor takes its cue from the UML Profiles syntax. With these editors, modelers may easily create stereotypes containing tagged values and set up inheritance relationships between stereotypes and extension relationships to metaclasses of arbitrary DSML's metamodels. Metaclasses may be imported using a custom popup menu entry, which shows when right-clicking the canvas of the editor, and are visualized using the graphical notation from Ecore.

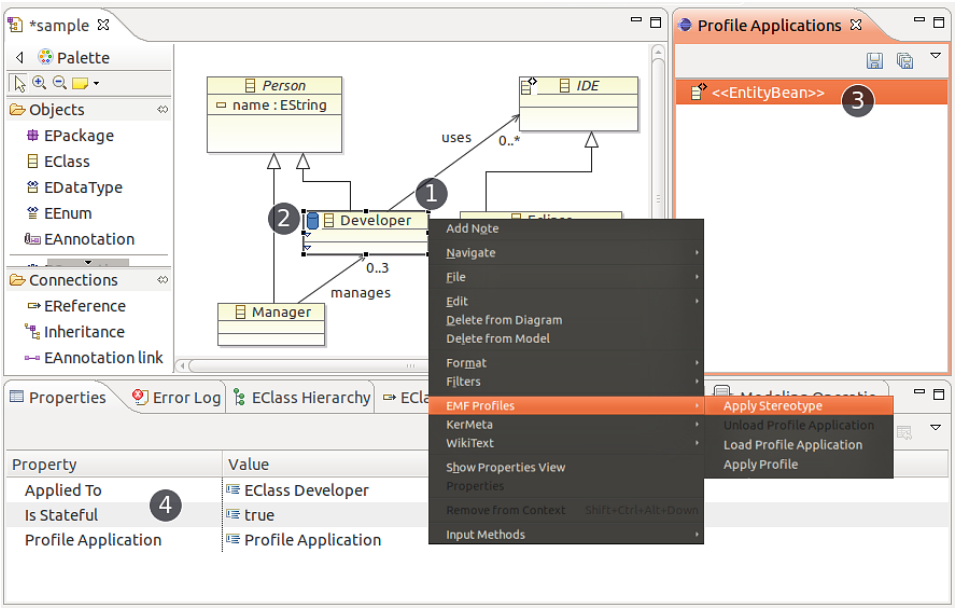

Figure 13 – EJB Profile Applied to Ecore Instance

Profile Application. Defined profiles may also be applied using any EMF-generated tree-based editor or any GMF-based diagram editor. The screenshot depicted in Fig. 13, shows the afore presented EJB profile applied to an example Ecore diagram. To apply profiles, our plug-in contributes a popup menu entry (cf. Fig. 13 (1)) which appears whenever a model element is right-clicked. With this menu, users may apply defined profiles (i.e., creating new profile application) or load already existing profile applications. Once a profile application is created or loaded, stereotypes may be applied using the same popup menu. When a stereotype is applied, the defined stereotype icon is attached to the model element (cf. Fig. 13 (2)). For this purpose we used the GMF Decoration Service, which allows to annotate any existing shapes by adding an image at a pre-defined location. Furthermore, we created a Profile Applications view, which shows all applied stereotypes of the currently selected model element (cf. Fig. 13 (3)). The currently selected model element is retrieved using the `ISelectionProvider` interface which is implemented by every EMF or GMF-based editor. For assigning the tagged values of an applied stereotype, we leverage the `PropertyView` (cf. Fig. 13 (4)) which generically derives all defined tagged values from the loaded profile's metamodel. The separate file resource which contains the profile applications is added to the `EditingDomain` of the modeling editor. Hence, as soon as the model is saved, all profile applications are saved as well. Finally, profile applications can be unloaded and reloaded at any time without loosing the application information.

```
1  // Load the model to be annotated
2  Resource modelResource = ...;
3  Entity anEntity = ...;
4
5  // Load the profile
6  Resource profileResource = ...;
7  Profile ejbProfile = (Profile) profileResource.getContents().get(0);
8
9  // Create the resource that contains the profile application
10 Resource profileApplicationResource = ...;
11
12 // Initialize the profile facade
13 IProfileFacade profileFacade = new ProfileFacadeImpl();
14 profileFacade.setProfileApplicationResource(profileApplicationResource);
15 profileFacade.loadProfile(ejbProfile);
16
17 // Apply stereotype
18 Stereotype sessionBean = ejbProfile.getStereotype("SessionBean");
19 StereotypeApplication application = null;
20 if (profileFacade.isApplicable(sessionBean, anEntitiy)) {
21  application = profileFacade.apply(sessionBean, anEntity);
22 }
23
24 // Set tagged values
25 EStructuralFeature isStateful = sessionBean.getTaggedValue("isStateful");
26 profileFacade.setTaggedValue(application, isStateful, Boolean.TRUE);
27
28 // Query and remove stereotype applications
29 EList<StereotypeApplication> applications =
30   profileFacade.getAppliedStereotypes(sessionBean);
31 profileFacade.removeStereotypeApplication(applications.get(0));
```

Listing 2 – Example for Using the EMF Profiles API

Using the EMF Profiles API. Profile definitions may also be created programmatically using the application programming interface (API) that is shipped with EMF (cf. [SBPM08] for more information on the EMF API). The same is true for profile applications. However, to ease the programmatic application of profiles, EMF Profiles provides a dedicated API, alongside the graphical user interfaces presented above. In the following, we briefly elaborate the usage of this API in terms of an example. In this example, we show how the *EJB profile* (cf. Fig. 4) is programmatically applied to an *ER model*. The corresponding Java code is depicted in Listing 2. In line 2 and line 3 of this listing, the resource containing the model to be annotated and an exemplary instance of `Entity` is loaded, respectively. Before we may apply the *EJB profile*, we first have to load the resource containing the corresponding profile definition (cf. line 6) and retrieve the instance of `Profile` representing the *EJB profile* from this resource (cf. line 7). To ease the application of stereotypes, EMF Profiles provides the interface `IProfileFacade` and an implementation of this interface called `ProfileFacadeImpl`, which is instantiated in line 13. Next, we specify a resource to which the profile application shall be saved and load the profile to be applied using the facade (cf. line 14–15). Please note that users may load multiple profiles to be applied at the same time with one profile facade instance. Now, we are set up to apply stereotypes. Therefore, a specific stereotype is obtained from the profile as shown in line 18. Using the method `isApplicable(Stereotype, EObject)` of the facade, we

may check whether the stereotype is applicable to the specified model element (cf. line 20). If this is the case, we may use the facade to apply the stereotype using the method `apply(Stereotype, EObject)` as depicted in line 21. This method returns the created instance of `StereotypeApplication`. The facade may also be used to set tagged values in a stereotype application. Therefore, we first obtain the tagged value from the stereotype (cf. line 25) and assign a specific value for this tagged value at a stereotype application using the method `setTaggedValue(StereotypeApplication, EStructuralFeature, EObject)` (cf. line 26). Finally, the facade provides methods for querying and removing stereotype applications (cf. line 29–31).

EMF Profiles and EMF-based Technologies. Profiles are common metamodels and profile applications are common instances of this metamodel. Thus, EMF-based technologies may easily be used to process profile applications. However, profile applications conform to a different metamodel than the annotated model and stereotype applications refer to model elements that are located in a different model resource. Thus, the prerequisite to *directly* process models and applied profiles using an EMF-based technology is that the technology has to be capable of processing *multiple models* conforming to *multiple metamodels*. The Atlas Transformation Language (ATL) [JABK08] is such a technology that meets these prerequisites. In the following, we show how ATL can be used to process profile applications and, subsequently, we discuss some possibilities to also enable using EMF-based technologies that are not capable of processing multiple metamodels and models.

EMF Profiles and ATL. In the following, we discuss a small excerpt of a model transformation defined in ATL. Please note that for using UML Profiles in ATL transformations (i.e., retrieving/instantiating stereotype applications and retrieving/setting tagged values), method calls from ATL to a Java-based API are necessary which leads to verbose transformation code as reported by Wimmer and Seidl [WS09]. In contrast, the meta-level architecture of EMF Profiles allows for a native support of EMF profiles in ATL transformations. In particular, EMF profiles may be used in ATL transformations as any other pure EMF-based model.

The example transformation is a so-called *copier transformation* [TJF+09] for duplicating an Ecore model which is annotated with an EJB profile application. Of course, not only the base model is supposed to be copied by the transformation, but also the annotations. In ATL transformations, it is possible to define an arbitrary number of input models and output models. Thus, profile applications may be loaded/produced as additional input/output models, respectively. The input models and output models for the copier transformation are specified in the header definition introduced by the module keyword (cf. line 1-3 in Listing 3). In addition to the input/output models of type Ecore, an additional input/output model of type EJB is specified. These additional models conform to the EJB profile which can be used in ATL as any other standard EMF-based model. As the class `StereotypeApplication` is derived from the class `EClass` (cf. Fig. 2), instances of `Stereotypes` may be treated in ATL transformation like common `EClasses`. Please note that in order to link profile applications with model elements residing in the base model (i.e., setting the `appliedTo` reference), the option "allow for inter-model links" has to be activated in the ATL execution engine before executing the model transformation.

Although the copier transformation is one of the simplest model transformations, it illustrates the most important aspects of using EMF profiles in ATL transformations. First, the profile application for the input model is queried in the input pattern (intro-

```
1  module Ecore2Ejb;
2  create OM : Ecore, OM : EJB
3     from IM : Ecore, IM : EJB;
4
5  rule copyEClass {
6    from
7      class : Ecore!EClass,
8      bean : EJB!Bean (bean.appliedTo.includes(class))
9    to
10     class_c : Ecore!EClass (
11       name <- class.name
12     ),
13     bean_c : EJB!Bean (
14       appliedTo <- class_c,
15       capacity <- bean.capacity,
16       ...
17     )
18 }
19 ...
```

Listing 3 – Copier transformation for Ecore models with EBJ annotations (excerpt)

duced by the keyword `from`, cf. line 6 in Listing 3) of the `copyEClass` transformation rule. This input pattern matches not only for an element of type `EClass`, but also for an element of type `Bean`. Furthermore, the input pattern has a filter assigned which is needed to check if the matched `Bean` stereotype application has been applied on the matched `EClass`. Second, stereotype applications may be instantiated as for example needed to produce a corresponding `Bean` stereotype application in the target model for the copier transformation example. As can be seen in the output pattern (introduced by the keyword `to` in line 9 in Listing 3) of the `copyEClass` transformation rule, stereotypes are instantiated like standard metaclasses and the tagged values are set as features of common metaclasses. The output pattern element `class_c` which instantiates a metaclass is equally defined as the output pattern element `bean_c` which actually instantiates the stereotype `Bean`. This small example illustrates that, for transformation engineers, it makes no difference when an input model or output model is actually a profile application.

EMF Profiles and One-(meta)model-only Technologies. If EMF-based technologies are not capable of processing multiple metamodels and models, they may not be employed *directly*. However, there are still several ways to enable also using such technologies. Basically, we therefore have to generate a new metamodel, which combines the profile definition and the metamodel that is extended by the profile and translate the annotated model and the stereotype applications to the newly generated combined model. Ultimately, we obtain one model conforming to one metamodel, which now can be processed by any EMF-based technology. Admittedly, this approach has its disadvantages. Besides having to generate a new metamodel and translate models before they can be processed, depending on the use case, we also might have to worry about propagating changes among the generated model and the original model.

One very compelling alternative to the aforementioned technique is to employ *VirtualEMF* [CJC11] for this task. Using this technology it is possible to compose several models (e.g., the annotated model and the profile application model) into one virtual model; that is, a composed view on multiple models. This virtual model behaves just as a normal model. Thus, every EMF-based technology may be used to process it. Another major advantage of this technology is that the virtual model may

even be *modified* and the combined models are kept in sync.

6 Related Work

One alternative to profiles as an annotation mechanism is to use weaving models (e.g., by using Modelink[10] or the Atlas Model Weaver[11] [FBJ+05, VdCFM10]). Model weaving enables to compose different separated models, and thus, could be used to compose a core model with a concern-specific information model in a non-invasive manner. For instance, in [FGT+10] the Atlas Model Weaver has been applied to represent automatically generated annotations for models by using transformations. However, although weaving models are a powerful mechanism, manually annotating models with weaving models is counter-intuitive. Since this is not the intended purpose of weaving models, users cannot annotate models using their familiar environment such as a diagramming editor which graphically visualizes the core model. Current approaches only allow to create weaving models with specific tree-based editors in which there is no different visualization of the core model and the annotated information. Not least because of this, weaving models may quickly become very complex and challenging to manage.

Recently, Kolovos et al. [KRDM+10] presented an approach called *Model Decorations* addressing a very similar goal as EMF Profiles. Kolovos et al. proposed to attach (or "decorate") the additional information in terms of text fragments in GMF's *diagram notes*. To extract or inject the decorations from or into a model, hand-crafted model transformations are employed which translate the text fragments in the notes into a separate model and vice versa. Although their approach is very related to ours, there also are major differences. First, for enabling the decoration of a model, an extractor and injector transformation has to be manually developed which is not necessary with EMF Profiles. Second, since Kolovos et al. exploit GMF notes, only decorating GMF-based diagrams is possible. In contrast to our approach, models for which no GMF editor is available cannot be annotated. Third, the annotations are encoded in a textual format within the GMF notes. Consequently, typos or errors in these textual annotations cannot be automatically identified and reported while they are created by the user. Furthermore, users must be familiar with the textual syntax as well as the decoration's target metamodel (to which the extractor translates the decorations) to correctly annotate a model. In EMF Profiles, stereotypes may only be applied if they are actually applicable according to the profile definition and editing the tagged values is guided by a form-based property sheet. Consequently, invalid stereotype applications and tagged values can be largely avoided.

EMF Facet[12], a spin-off of the MoDisco subproject [BCJM10] of Eclipse, is another approach for non-intrusive extensions of Ecore-based metamodels. In particular, EMF Facet allows to define additional derived classes and features which are computed from already existing model elements by model queries expressed, e.g., in Java or OCL. Compared to EMF Profiles, EMF Facet targets on complementary extension direction, namely the dynamic extension of models with additional *transient information* derived from queries. In contrast, EMF Profiles allow to add new (not only derived) information and is able to persist this additional information in separate files. Nevertheless, the combination of both complementary approaches is worth to be subject for future work.

[10] `http://www.eclipse.org/gmt/epsilon/doc/modelink`
[11] `http://www.eclipse.org/gmt/amw`
[12] `http://www.eclipse.org/modeling/emft/facet`

For example, this would allow to automatically extend or complete models based on EMF Facet queries and persist this information with EMF Profiles.

The concept of meta-packages has been proposed in [CESW04] for the lightweight extension of the structural modeling language XCore which is based on packages, classes, and attributes. New modeling concepts are defined by extending the base elements of XCore and can be instantly used in the standard XCore editor. Compared to meta-packages, EMF Profiles are more generic, because not only one modeling language may be extended, but any Ecore-based modeling language.

Besides extensibility, we also discussed reusability aspects in this paper by introducing the notion of genericity for profiles. Genericity has been also subject for other investigations in the area of model-driven engineering. In [dLG10], genericity has been introduced for metamodels to define model simulators for a family of languages sharing the same semantics. Another kind of generic in-place transformations has been discussed in [MMBJ09]. In particular, the authors presented generic model refactorings which can be reused for different metamodels sharing similar modeling elements. Finally, generic out-place transformations have been proposed in [CGdL11] for the model transformation language ATL. In contrast to these works, we are not focusing in this paper on how to implement reusable transformations for generic profiles. Instead, our aim is to define generic profiles which can be bound to and used for concrete metamodels which have to fulfill some properties.

7 Conclusions and Future Work

In this paper, we adapted the notion of UML Profiles to the realm of DSMLs residing in the Eclipse Modeling Framework. Using our prototype EMF Profiles, DSMLs may be extended easily in a non-invasive manner by defining profiles in the same way as done in UML tools. Therefore, we surveyed the UML standard concerning the profile mechanism, as well as existing UML modeling tools. The results of this survey acted as input for designing and developing EMF Profiles. The resulting design decisions in EMF Profiles are clearly in line with the UML standard; however, some constraints of UML profiles have been omitted consciously to increase the expressive power of profiles. Moreover, we introduced two novel mechanisms, namely *generic profiles* and *meta profiles*, for reusing defined profiles with several DSMLs. Although, the presented approach has been presented based on EMF, the general procedure is also applicable for other metamodeling frameworks which comprise a similar metalevel architecture as EMF. Furthermore, the presented metalevel lifting strategies may also be adopted for other scenarios in which model elements on M_1 need to be instantiated.

We applied EMF Profiles successfully in the context of our model versioning system AMOR[13]. In particular, we created and applied a *change profile* for annotating changes performed on models. Moreover, we also used EMF Profiles for marking conflicts caused by concurrent changes of the same model artifact using a *conflict profile*. Both profiles have been defined as *meta profiles* to build change detection and conflict detection components, which are generically applicable to all EMF-based modeling languages.

In the future, we plan to elaborate on more sophisticated restriction mechanisms to allow constraining the application of stereotypes (e.g. with OCL conditions) and composing several independent profiles which are not mutually complementary in one profile application as proposed by [NGTS10]. A consistent mix of several profiles

[13] `http://www.modelversioning.org`

requires a mechanism to specify conditions constraining applicability across more than one profile. For instance, one may need to specify that a stereotype of profile A may only be applied after a stereotype of profile B, holding a specific tagged value, has been applied. Next, we plan to implement code generators for deriving a profile-specific application programming interfaces (API) allowing to create, modify, and access applications of a profile. So far, we only provide a generic API, which offers no static type checking and which might be not as convenient to use as a dedicated API generated for a specific profile. Furthermore, we aim at integrating EMF Profiles with the EMF Facet project to combine their complementary features. Thereby, a synergy of the extension mechanism of EMF Profiles for additional persisted information and of EMF Facet's for derived information can be accomplished.

Last but not least, we want to explore how EMF Profiles fit into the scope of the *Metamodel Extension Facility* request for proposal [OMGb] recently published by the OMG.

References

[AK01] C. Atkinson and T. Kühne. The Essence of Multilevel Metamodeling. In *Proceedings of the International Conference on the Unified Modeling Language (UML'01)*, volume 2185 of *LNCS*, pages 19–33. Springer, 2001. doi:10.1007/3-540-45441-1_3.

[AK07] C. Atkinson and T. Kühne. A Tour of Language Customization Concepts. *Advances in Computers*, 70:105–161, 2007. doi:10.1016/S0065-2458(06)70003-1.

[AKHS03] C. Atkinson, T. Kühne, and B. Henderson-Sellers. Systematic stereotype usage. *Software and System Modeling*, 2(3):153–163, 2003. doi:10.1007/s10270-003-0027-9.

[BCJM10] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot. MoDisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the International Conference on Automated Software Engineering (ASE'10)*, pages 173–174. ACM, 2010. doi:10.1145/1858996.1859032.

[BLS+09] Petra Brosch, Philip Langer, Martina Seidl, Konrad Wieland, Manuel Wimmer, Gerti Kappel, Werner Retschitzegger, and Wieland Schwinger. An Example Is Worth a Thousand Words: Composite Operation Modeling By-Example. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, volume 5795 of *LNCS*, pages 271–285. Springer, 2009. doi:10.1007/978-3-642-04425-0_20.

[CESW04] T. Clark, A. Evans, P. Sammut, and J. Willans. *Applied Metamodelling, A Foundation for Language Driven Development.* 2004. URL: http://www.ceteva.com.

[CGdL11] Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. Generic model transformations: *Write Once, Reuse Everywhere.* In *Proceedings of the 4th International Conference on Theory and Practice of Model Transformations (ICMT'11)*, volume 6707 of *LNCS*, pages 62–77. Springer, 2011. doi:10.1007/978-3-642-21732-6_5.

[Che76] P.P. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1:9–36, 1976. doi:10.1145/320434.320440.

[CJC11] C. Clasen, F. Jouault, and J. Cabot. VirtualEMF: A Model Virtualization Tool. In *Advances in Conceptual Modeling, Recent Developments and New Directions*, volume 6999 of *LNCS*, pages 332–335. Springer, 2011. doi:10.1007/978-3-642-24574-9_43.

[DDZ08] Jürgen Dingel, Zinovy Diskin, and Alanna Zito. Understanding and improving UML package merge. *Software and System Modeling*, 7(4):443–467, 2008. doi:10.1007/s10270-007-0073-9.

[dLG10] Juan de Lara and Esther Guerra. Generic meta-modelling with concepts, templates and mixin layers. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MODELS'10)*, volume 6394 of *LNCS*, pages 16–30. Springer, 2010. doi:10.1007/978-3-642-16145-2_2.

[FBJ+05] M. Didonet Del Fabro, J. Bézivin, F. Jouault, E. Breton, and G. Gueltas. AMW: A Generic Model Weaver. In *Journée sur l'Ingénierie Dirigée par les Modéles (IDM'05)*, 2005.

[FGT+10] M. Fritzsche, W. Gilani, M. Thiele, I. Spence, T.J. Brown, and P. Kilpatrick. A scalable approach to annotate arbitrary modelling languages. In *Proceedings of Modellierung 2010*, volume 161 of *LNI*, pages 301–318. GI, 2010.

[JABK08] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. ATL: A Model Transformation Tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008. doi:10.1016/j.scico.2007.08.002.

[KRDM+10] D. Kolovos, L. Rose, N. Drivalos Matragkas, R. Paige, F. Polack, and K. Fernandes. Constructing and Navigating Non-invasive Model Decorations. In *Proceedings of the International Conference on Theory and Practice of Model Transformations (ICMT'10)*, volume 6142 of *LNCS*, pages 138–152. Springer, 2010. doi:10.1007/978-3-642-13688-7_10.

[KT08] S. Kelly and J.-P. Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Press, 2008. doi:10.1002/9780470249260.

[Küh06] T. Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5:369–385, 2006. doi:10.1007/s10270-006-0017-9.

[LWWC11] Philip Langer, Konrad Wieland, Manuel Wimmer, and Jordi Cabot. From UML Profiles to EMF Profiles and Beyond. In *Proceedings of the 49th International Conference on Objects, Models, Components, Patterns (TOOLS'11)*, volume 6705 of *LNCS*, pages 52–67. Springer, 2011. doi:10.1007/978-3-642-21952-8_6.

[MMBJ09] Naouel Moha, Vincent Mahé, Olivier Barais, and Jean-Marc Jézéquel. Generic model refactorings. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09)*, volume 5795 of *LNCS*, pages 628–643. Springer, 2009. doi:10.1007/978-3-642-04425-0_50.

[MS89] D. Musser and A. Stepanov. Generic Programming. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC'88)*, volume 358 of *LNCS*, pages 13–25. Springer, 1989.

[NGTS10] F. Noyrit, S. Gerard, F. Terrier, and B. Selic. Consistent Modeling Using Multiple UML Profiles. In *Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS'10)*, volume 6394 of *LNCS*, pages 392–406. Springer, 2010. doi:10.1007/978-3-642-16145-2_27.

[OMGa] Object Management Group OMG. Meta Object Facility, Version 2.0, http://www.omg.org/spec/MOF/2.0/PDF/ (2006).

[OMGb] Object Management Group OMG. Metamodel Extension Facility - Request for Proposal, http://www.omg.org/cgi-bin/doc?ad/11-06-22.pdf (2011).

[OMGc] Object Management Group OMG. Unified Modeling Language Infrastructure Specification, Version 2.1.2, http://www.omg.org/spec/UML/2.1.2/Infrastructure/PDF (2007).

[SBPM08] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *Eclipse Modeling Framework*. Eclipse Series. Addison-Wesley Professional, 2nd edition, 2008.

[Sel07] B. Selic. A Systematic Approach to Domain-Specific Language Design Using UML. In *Proceedings of the 10th International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'07)*, pages 2–9. IEEE Computer Society, 2007. doi:10.1109/ISORC.2007.10.

[TJF+09] M. Tisi, F. Jouault, P. Fraternali, S. Ceri, and J. Bézivin. On the Use of Higher-Order Model Transformations. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA'09)*, volume 5562 of *LNCS*, pages 18–33. Springer, 2009. doi:10.1007/978-3-642-02674-4_3.

[VdCFM10] Juan M. Vara, Valeria de Castro, Marcos Didonet Del Fabro, and Esperanza Marcos. Using weaving models to automate model-driven web engineering proposals. *IJCAT*, 39(4):245–252, 2010. doi:10.1504/IJCAT.2010.036028.

[WS09] M. Wimmer and M. Seidl. On Using UML Profiles in ATL Transformations. In *Proceedings of the 1st International Workshop on Model Transformation with ATL (MtATL'09)*, 2009.

About the authors

Philip Langer is postdoctoral researcher in the Business Informatics Group at the Vienna University of Technology. He received a PhD in Computer Science from the Vienna University of Technology in 2011 for his thesis on model versioning and model transformation by demonstration. His current research is focused on model evolution, model transformations, and model execution in the context of model-driven engineering. For further information about his research activities, please visit `http://www.big.tuwien.ac.at/staff/planger` or contact him at `langer@big.tuwien.ac.at`.

Konrad Wieland is technical consultant at LieberLieber GmbH. He studied Business Informatics at the Vienna University of Technology from 2003 to 2009. From 2009 to 2011, he worked as research assistant at the Business Informatics Group (Vienna University of Technology) in the area of model versioning and collaborative conflict resolution. He received a PhD in Computer Science from the Vienna University of Technology in 2011. Konrad can be reached at `konrad.wieland@lieberlieber.com`.

Manuel Wimmer is working as a post-doc researcher at the Business Informatics Group of the Vienna University of Technology. His research interests comprise Web engineering and model engineering; in particular model transformations based on formal methods, generating transformations by-example as well as applying model transformations to deal with model (co-)evolution. Currently, he is on leave working as visiting researcher at the Software Engineering Group of the University of Málaga (Spain). For further information about his research activities, please visit `http://www.big.tuwien.ac.at/staff/mwimmer` or contact him at `wimmer@big.tuwien.ac.at`.

Jordi Cabot is currently leading the AtlanMod team, an INRIA research group at École des Mines de Nantes (France). Previously, he has been a post-doctoral fellow at the University of Toronto, a senior lecturer at the UOC (Open University of Catalonia) and a visiting scholar at the Politecnico di Milano. He received the BSc and PhD degrees in Computer Science from the Technical University of Catalonia. His research interests include conceptual modeling, model-driven and web engineering, formal verification and social aspects of software engineering. He has written more than 70 publications in international journals and conferences in the area. Apart from his scientific publications, he writes and blogs about all these topics in his Modeling Languages portal (`http://modeling-languages.com`).

Acknowledgments We would like to thank *Hugo Brunelière* for our continuous collaboration and his kind support of this work at several occasions. Furthermore, we are very grateful to *Tanja Mayerhofer* for her valuable feedback and suggestions for improvement of this paper.

This work has been partially funded by the Austrian Science Fund (FWF) under grant J 3159-N23 and by the European CESAR ARTEMIS JOINT UNDERTAKING project.