From UML Profiles to EMF Profiles and Beyond[⋆]

Philip Langer[1], Konrad Wieland[2], Manuel Wimmer[2], and Jordi Cabot[3]

[1] Johannes Kepler University Linz, Austria
philip.langer@jku.at
[2] Vienna University of Technology, Austria
{wieland|wimmer}@big.tuwien.ac.at
[3] INRIA & Ecole des Mines de Nantes, France
jordi.cabot@inria.fr

Abstract. Domain-Specific Modeling Languages (DSMLs) are getting more and more attention as a key element of Model Driven Engineering. As any other software artefact, 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 its 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 adaptation scenarios. We believe a similar concept for DSMLs would provide an easier 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

Accepted at TOOLS 2011 Draft

1 Introduction

Domain-Specific Modeling Languages (DSMLs) have gained much attention in the last decade [7]. 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

[⋆] This work has been partly funded by the Austrian Federal Ministry of Transport, Innovation and Technology (BMVIT) and FFG under grant FIT-IT-819584.

all related artifacts as well as all DSML-specific components of the modeling environment have to be re-created or adapted.

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 [**?**]. 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 burdens 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 annotation mechanism for *existing* DSMLs. We believe the usage of profiles in the realm of DSMLs brings several benefits:

(1) 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.

(2) 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 further be annotated with several stereotypes (even contained in different profiles) at the same time which is equivalent to the model element having multiple types [2]. Furthermore, the additional information introduced by the profile application is kept separated from the model and, therefore, does not pollute the actual model instances.

(3) 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.

(4) 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

Accepted at TOOLS 2011 Draft

[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`)

technologies to 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, the notion of profiles has not been adopted in current metamodeling tools. Thus, the contribution of this paper is to adapt the notion of UML profiles to arbitrary modeling languages residing in the Eclipse Modeling Framework[3] (EMF) which is currently one of the most popular metamodeling frameworks. Thanks to this, existing modeling languages may easily be extended by profiles in the same way as it is known from UML tools. Besides this, 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.

2 From UML Profiles to EMF Profiles

In this section, we present the *standard profile* mechanism (as known from UML) for EMF. Firstly, we disclose our design principles. Secondly, we discuss how the profile mechanism may be integrated in EMF in a way that profiles can seamlessly be used within EMF following the previous 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.

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

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

Draft
Accepted at TOOLS 2011

specifying UML Profiles, resides, in terms of the metamodeling stack [9], at the meta-metalevel M_3 [13] 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.

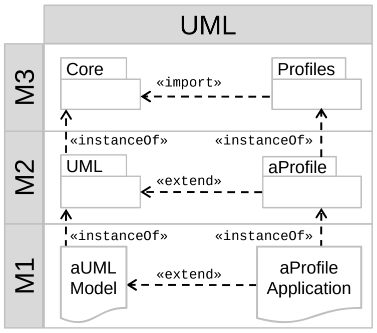


Fig. 1: UML Architecture

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 [12]) 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 [1].

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 the first column 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

Accepted at TOOLS 2011 Draft

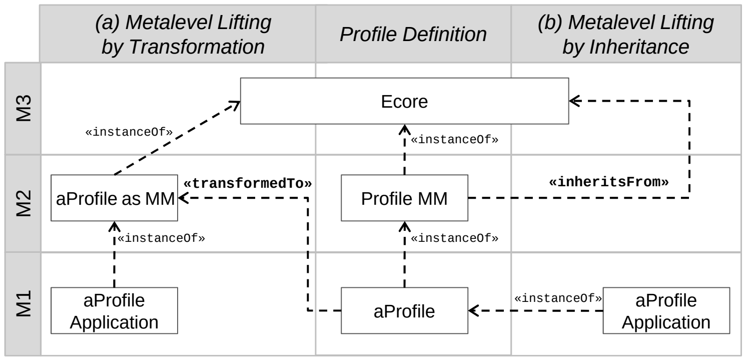


Fig. 2: EMF Profile Architecture Strategies

Draft
Accepted at TOOLS 2011

`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 the right column 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.

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 profile and profile application 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 the profile definition language is illustrated in package *Standard EMF Profile* of Fig. 3. As a positive side effect of choosing the metalevel lifting strategy 2, the class `Stereotype` may contain, as an `EClass`, also

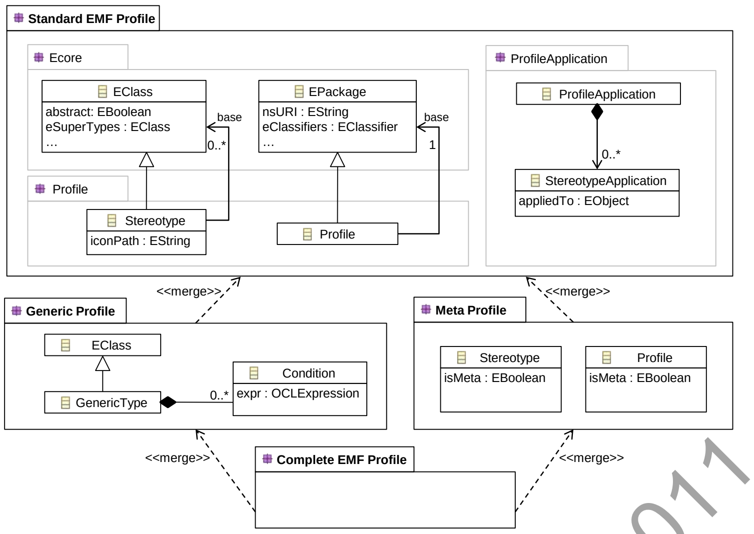


Fig. 3: EMF Profile Metamodel

Accepted at TOOLS 2011 Draft

`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. Please note that stereotype applications also 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 (instance of the *Profile* package) is saved, we automatically add `StereotypeApplication` as a superclass to each specified stereotype. To recall, this is possible because each `Stereotype` is an `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 3.

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 [4]. Fig. 4(a) depicts an excerpt of the ER metamodel and the EJB profile. The EJB profile contains the stereotypes `SessionBean` and `EntityBean`, which both extend the metaclass `Entity` of the ER metamodel. Besides, the profile

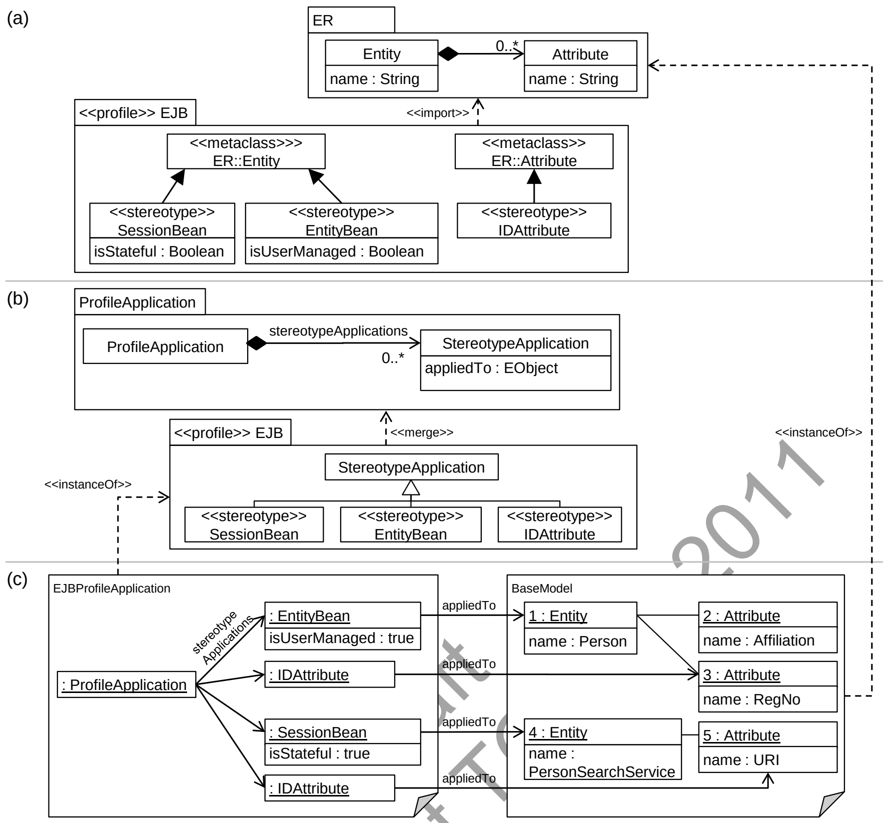


Fig. 4: EMF Profiles by Example: (a) Profile definition user-view, (b) Internal profile representation, (c) Profile application

Accepted at TOOLS 2011

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 `ProfileApplication` 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 Going Beyond UML Profiles

Originally, the profile mechanism has been specifically developed for UML. Hence, profiles may only extend the UML metamodel. In the previous section, 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 *specific set of DSMLs*. Being aware of these specific DSMLs' metamodels, the user wants to take control of the applicability of stereotypes to a manually selected set of metaclasses.

(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.

To tackle 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 tackling scenario (2), we propose *meta profiles* which may immediately be applied to all DSMLs implemented by an Ecore-based metamodels.

Accepted at TOOLS 2011
Draft

3.1 Generic Profiles

The goal behind generic profiles is to reuse a profile specification for several "user-selected" DSMLs. Therefore, a profile should not dependent on a specific metamodel. Inspired by the concepts of generic programming [10], 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.

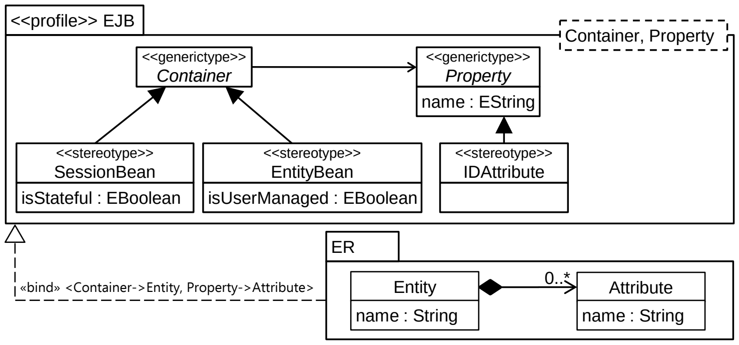


Fig. 5: Generic EJB Profile and its Binding to the ER metamodel

Accepted at TOOLS 2011 Draft

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` and the stereotype `IDAttribute` extends `Attributes` (cf. Fig. 4). To turn this concrete profile into a generic one, we now use two generic types, named `Container` and `Property` in Fig. 5, 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, `Entities` can be annotated with the stereotype `EntityBean`. For marking the `Entity`'s ID attribute, the EJB profile introduces the stereotype `IDAttribute` which extends `Attributes`. This is reasonable, because we are aware of the fact that `Entities` *contain* `Attributes` 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 conditions to generic profiles. Such conditions are specified by simply adding references or attributes to generic types. This is possible because, as a subclass of `EClass`, generic types may contain `EReferences` and `EAttributes`. By adding such a reference or attribute in a generic type, a profile designer states that there must be a corresponding reference or attribute in the metaclass which is bound to the generic type. Internally, these references and attributes are translated to OCL constraints which are evaluated in the context of the metaclass a user intends to bind. Furthermore, the profile designer must specify which meta-features, such as the cardinality of the reference or attribute in a generic type, shall be enforced. In our example in Fig. 5, the profile designer specified a reference from the generic type `Container` to `Property` as well as an attribute `name` in `Property`. To enforce this, the OCL constraints in Listing 1.1 are generated. These constraints must be satisfied by each metamodel on which we want to apply this profile on.

Listing 1.1: OCL Constraints generated for `Container` and `Property`.

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

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. 5, the generic types `Container` and `Property` are bound to the metaclasses in the ER metamodel `Entity` and `Attribute`, 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.

Accepted at TOOLS 2011 Draft

3.2 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).

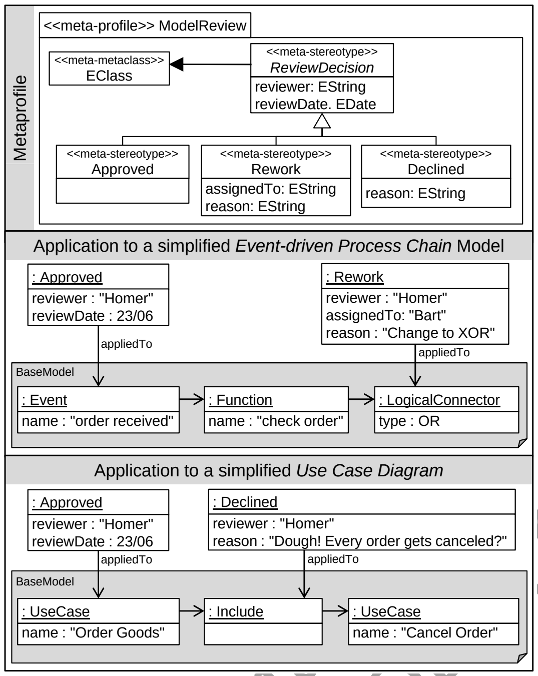


Fig. 6: Meta profile Example: The Model Review Profile

Draft
Accepted at TOOLS 2011

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 an instance of an instance of `EClass`). This approach is inspired by the concept of potency known from multilevel metamodeling [1]. 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 also be configured to be instantiable on the level $n - p$ instead of the next lower level. 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 regarded 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. 6). 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. 6). 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. 6, 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 ≪*Approve*≫ 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 can also be applied to any other modeling language. Of course, also UML itself is supported by EMF Profiles. Therefore, the model review profile may also be applied to, for example, a UML Use Case Diagram (cf. Fig. 6). In this figure, the stereotype ≪*Approve*≫ has been assigned to the `UseCase` named "Order Goods" and the stereotype ≪*Declined*≫ is applied to the `Includes` relationship.

3.3 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.

4 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 and Graphical Modeling Framework[4] (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[5].

[4] `http://www.eclipse.org/gmf`
[5] `http://www.modelversioning.org/emf-profiles`

Draft Accepted at TOOLS 2011

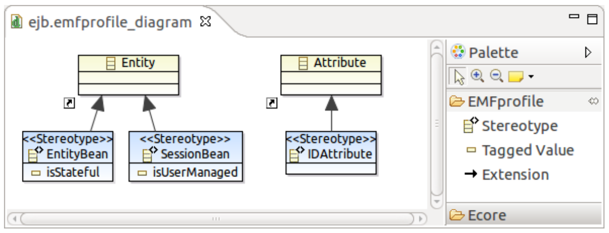
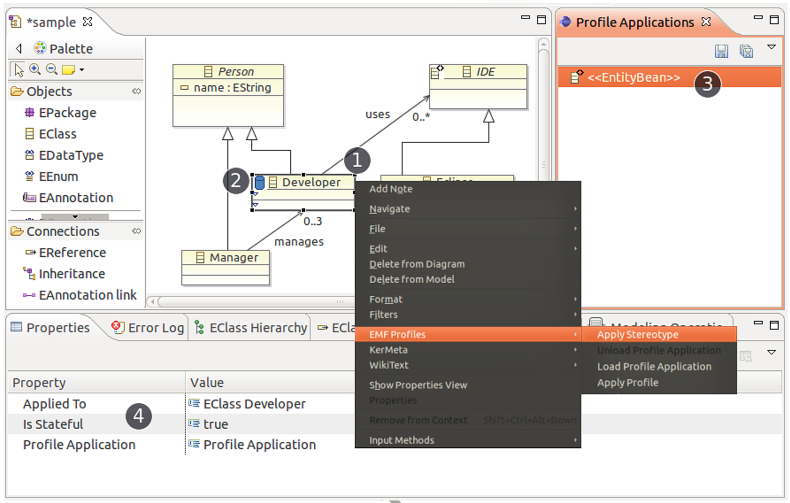


Fig. 7: EJB Profile Defined with Graphical EMF Profiles Editor



Property	Value
Applied To	EClass Developer
Is Stateful	true
Profile Application	Profile Application

Fig. 8: Screenshot of Applied EJB Profile to an Ecore Diagram

Accepted at

Profile Definition. To define a profile, modelers may apply either the tree editor automatically generated from the Profile Metamodel or our graphical EMF Profiles Editor which is realized with GMF (cf. Fig. 7 for a screenshot). The graphical 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 by a custom popup menu entries when right-clicking the canvas of the editor and are visualized using the graphical notation from Ecore.

Profile Application. Defined profiles may also be applied using any EMF-generated tree-based editor or any GMF-based diagramm editor. The screenshot depicted in Fig. 8, shows the afore presented EJB profile applied to an example Ecore diagram. To apply profiles, our plugin contributes a popup menu entry (cf. Fig. 8 (1)) which appears whenever a model element is right-clicked. By this menu, users may apply defined profiles (i.e., creating new profile application) or import already existing profile applications. Once a profile application is created or imported, 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. 8 (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. 8 (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. 8 (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.

5 Related Work

One alternative to profiles as an annotation mechanism is to use weaving models (e.g., by using Modelink[6] or the Atlas Model Weaver[7] [5]). 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. However, although weaving models are a powerful mechanism, 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. presented an approach called *Model Decorations* [8] tackling 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, handcrafted 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

Accepted at TOOLS 2011
Draft

[6] `http://www.eclipse.org/gmt/epsilon/doc/modelink`
[7] `http://www.eclipse.org/gmt/amw`

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[8], a spin-off of the MoDisco subproject [3] 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 seems to be a subject for future work. For example, this would allow to automatically extend or complete models based on EMF Facet queries and persist this information with EMF Profiles.

6 Conclusions and Future Work

Accepted at TOOLS 2011 Draft

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 easily extended in a non-invasive manner by defining profiles in the same way as done in UML tools. 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 successfully applied EMF Profiles for instance in the context of our model versioning system AMOR[9]. In AMOR 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, i.e., independent of the used 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 com-

[8] `http://www.eclipse.org/modeling/emft/facet`
[9] `http://www.modelversioning.org`

plementary in one profile application as proposed by [11]. A consistent mix of several profiles 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. EMF‘s OCL plug-in and EMF Query[10] need to be extended to cope with this inter-profile scenarios. Next, we plan to derive an easy-to-use API for programmatically creating, modifying, and accessing profile applications. Finally, we aim at integrating EMF Profiles into the EMF Facet project to combine their complementary features. By this, a synergy of the extension mechanism of EMF Profiles for additional persisted information and of EMF Facet‘s for derived information can be accomplished.

References

1. Atkinson, C., Kühne, T.: The Essence of Multilevel Metamodeling. In: UML 2001– The Unified Modeling Language. pp. 19–33. Springer (2001)
2. Atkinson, C., Kühne, T.: A Tour of Language Customization Concepts. Advances in Computers 70, 105–161 (2007)
3. Bruneliere, H., Cabot, J., Jouault, F., Madiot, F.: MoDisco: a generic and extensible framework for model driven reverse engineering. In: Proceedings of the 25th International Conference on Automated Software Engineering (ASE'10). pp. 173–174. ACM (2010)
4. Chen, P.P.S.: The Entity-Relationship Model—Toward a Unified View of Data. ACM Transactions on Database Systems 1, 9–36 (1976)
5. Fabro, M.D.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: a generic model weaver. In: Proceedings of the 1re Journe sur l'Ingnierie Dirige par les Modles (IDM'05), France (2005)
6. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press (2008)
7. Kolovos, D., Rose, L., Drivalos Matragkas, N., Paige, R., Polack, F., Fernandes, K.: Constructing and Navigating Non-invasive Model Decorations. In: Theory and Practice of Model Transformations (ICMT'10). pp. 138–152. Springer (2010)
8. Kühne, T.: Matters of (meta-) modeling. Software and Systems Modeling 5, 369–385 (2006)
9. Musser, D., Stepanov, A.: Generic Programming. In: Symbolic and Algebraic Computation. LNCS, vol. 358, pp. 13–25. Springer (1989)
10. Noyrit, F., Gerard, S., Terrier, F., Selic, B.: Consistent Modeling Using Multiple UML Profiles. In: Model Driven Engineering Languages and Systems (MoDELS‘10). pp. 392–406. Springer (2010)
11. Object Management Group (OMG): Meta Object Facility, Version 2.0, `http://www.omg.org/spec/MOF/2.0/PDF/` (2006)
12. 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)
13. Selic, B.: A Systematic Approach to Domain-Specific Language Design Using UML. Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on pp. 2–9 (2007)

Accepted at TOOLS 2011 Draft

[10] `http://www.eclipse.org/modeling/emf/?project=query`