

Virtual Extension of Meta-models with Facet Tools

Jonathan Pepin^{1,2}, Pascal André¹, Christian Attiogbé¹ and Erwan Breton²

¹*AeLoS Team LS2N CNRS UMR 6004, University of Nantes, France*

²*Mia-Software - Nantes, 44324 Nantes cedex 3, France*

Keywords: Meta-model, Weaving, Mapping, EMF, Facet, Transformation Tools.

Abstract: MDE emphasizes the use of models and meta-models to improve the software productivity and some aspects of the software quality such as maintainability or interoperability. In the software industry, Model Driven Engineering (MDE) techniques have proven useful not only for developing new software applications but for re-engineering legacy systems. However the stakeholders have to face costly maintenance operations due to frequent new standards and upgraded releases of software modules they depend on. Therefore, due to the limitations of existing techniques, solutions ensuring a better adaptability and flexibility of model evolution tools are needed. We propose an improved technique of virtual extension of meta-models with Facets that enables one to modify meta-models already in use without rebuilding completely the software product. This technique has been implemented and experimented for model alignment and evolution.

1 INTRODUCTION

The importance of modelling in software engineering and the popularity of the UML notation result in the dissemination of model-based tools and the emergence of a modelling language industry based on the model-driven approach (MDA). The Meta-Object Facility (MOF) language is then the foundation for an ecosystem of Domain Specific Languages (DSL) (Brambilla et al., 2012). Some of them are standards like BPMN, BMM, SoaML, SysML, UPDM while others are specific. Model Driven Engineering (MDE) emphasizes the use of models and meta-models to improve the software productivity and some aspects of the software quality such as maintainability or interoperability. MDE techniques have proven useful not only for developing new software applications but for re-engineering legacy systems and dynamically configuring running systems (Cuadrado et al., 2014).

In the context of software maintenance, including model-driven reverse engineering of a legacy system, we face challenges related to model evolution, model composition, multi-generation models and multi-layered models.

- Meta-models are essential in MDE since many model processing are described at the meta-model level through transformation rules or operations. However this is also a pitfall since both models and meta-models evolve separately. In particu-

lar, the modelling languages depend on standards that evolve continuously and the related models must also evolve: there is a dependency chain to maintain. **Model evolution** is even more complex when (meta-)models are parts of larger models.

- As mentioned by El Kouhen in (El Kouhen, 2016), MDE promotes the separation of concerns to deal with the complexity and maintainability of software design. This current practice implies the creation of several heterogeneous models using different notations (and therefore meta-models). Since the semantics of each individual model is limited, the consistency and completeness of the individual models are easier to prove but the difficulty is delegated to the composition of these individual models. *Model composition* is the symmetric paradigm of separation of concerns (Atlee et al., 2007). It is then necessary to compose models to reason on the overall designed system for many purposes such as: checking the global consistency of the models, understanding the interactions between models, generating code, etc. In particular, we are interested in compositions that do not interfere with the individual models.
- Multi-generation occurs when different releases of one model (or meta-model) are maintained *e.g.* for different customers or different branches of a company. This may occur also in software product lines or software customization.

- Multi-layering occurs when we need to maintain links between models at different levels of abstraction (traceability, refinement...). Examples are the CIM-PIM-PSM stack or the enterprise architecture alignment stack (Clark et al., 2011).

A *typical scenario* for a software provider is to deliver customised release to some customers but customisation leads to a tricky maintenance problem when models and meta-models evolve. This includes the case of legacy code built using various DSL with various development methods at different levels of abstraction (from implementation code up to business processes). In order to represent specific (or customised) aspects we extend meta-models with new concepts, attributes and relations. It is always possible to create the new meta-model with references to the initial meta-model but any modification requires to rebuild the delivery product. This is also the case when the modelling language evolves; the new release requires tool adaptation.

In all these cases of software maintenance, including model-driven reverse engineering of a legacy system, it is a primary requirement to have techniques to build new models without modifying the source models (to preserve source property), we call it **non-intrusive model mapping**. The existing approaches are not fully satisfying because they are either intrusive, volatile or too generic. Paige et al. mentioned several challenges of evolving models in MDE (Paige et al., 2016). The work presented in this paper is a practical contribution to the *dependency heterogeneity* challenge. We focus on model mappings that support several semantic links, models (or meta-models) evolution and persistence, while staying non-intrusive by preserving their parts. More precisely our contribution is threefold:

- A mapping definition that enables to connect models without breaking their legacy semantics and without rebuilding the associated software tool support.
- A mapping technique which is compliant with the existing MDA tools. Taking an industrial point of view, the question is not only to find a mapping meta-model, which is usually depending on the context (the models we work with), but also to implement mapping techniques which are compliant with the existing MDA tools and efficient for large-scale applications.
- A mapping tool that can be reused by others and customised to similar model transformation in practice. At the implementation level, maintaining a link between model elements can also be seen as an *object relation mapping* (ORM) problem (Ambler, 1997) preserving the link multiplici-

ties and a bi-directional navigation. This problem needs a robust algorithm engine to maintain the constraints imposed by the multiplicities.

The paper mainly targets a tool set; it is structured as follows. In Section 2 we review model mapping techniques and motivate our choices. Section 3 overviews EMF Facet and introduces our improvements. Persistence, navigation and testing issues are discussed in Section 4. New user interface facilities are presented on the illustrating example in Section 5. The application field is detailed in Section 6 including a tutorial, user stories and a return on experience of larger case studies. Finally, Section 7 summarises the contribution and draws open perspectives.

2 BACKGROUND AND REQUIREMENTS

In this section we overview the basic requirements, define the core concepts and compare with related works to set the contribution requirements.

2.1 Background and Core Concepts

A software system sustains several releases that can even co-exist for different users with different hardware. The definition of models needs enhancements: new attributes, new entities, new classifications, new links, ... Thus the real life systems generally handle more than one model that may co-exist with different semantics *i.e.* as defined by their meta-models. To capture this reality, we need a particular *model mapping* technique that owns the following criteria:

1. **Non-intrusiveness:** the mapping must not modify the individual models because they evolve independently.
2. **Semantics:** the mapping is not simply a set of links, it supports a semantic relation to connect differently the concepts with an equivalence class of interpretation (an ontology of the concepts and links).
3. **Link Resolution:** the mapping techniques must provide a mechanism to navigate by mapping links directly from the source model to the target model and reciprocally.
4. **Serialization:** the mapping links must be persistent to store the working environment.

These properties come from lessons learned during model maintenance activities. This was the basis for finding an adequate "mapping semantics".

Model mapping is one of the "*Model manipulation and management challenges*" of (France and Rumpe,

2007) and in particular to the points (2) and (3) mentioned by its authors: (2) *maintaining traceability links among model elements to support model evolution and round trip engineering* and (3) *maintaining consistency among viewpoints*. Model mapping techniques are useful for model composition, decomposition or synchronization (France and Rumpe, 2007). In fact there are many "similar" operations and our first goal was to find the adequate semantics and an associated operational technique. Clavreul identified 88 model composition techniques for different purposes in his systematic review (Clavreul, 2011). Model Mapping is a model composition that preserves the components. It is called the "*model-based correspondence*" by Clavreul. Considering our requirements, we reduce the field to *model mapping* and we retain five mapping techniques: extension, merging, annotation, weaving and facets.

Model Extension or Merging. In the first approach, the meta-model of one layer is extended with the concepts of another one. In the second approach, the meta-models are merged in a single big model. In both case, the meta-models loose their consistency and they can hardly evolve (flexibility loss). Clavreul's mapping language is a merging approach and the architects must learn a DSL. We opted for a mechanized approach by providing a simple tool which applies at both compile and run time. El Kouhen (El Kouhen, 2016) proposed an unified methodology to compose models based on meta-model extensions. The composition operators are symmetric (commutative *e.g.* merge, parallel) or asymmetric (weaving in the meaning of AOP, sequential integration). His work is largely inspired by (Marchand et al., 2012) who gave a formal semantics for weaving and merging through morphisms of a category theory. Our approach can be seen as an *ad hoc* model composition in their classification since our mapping uses semantic information of the source models while their approach is a model mapping in our classification. Their mapping approaches enable the separation of concerns, but the merging and weaving do not preserve the legacy models (and the existing related tool support) because their result is a new model. Model merging or extension are intrusive techniques because the source models disappear in the target model.

Model Annotation and Weaving. Model mapping is close to model weaving as defined by (Jouault et al., 2010) which was inspired by Aspect Oriented Modelling. "*Model weaving operations are performed between two or more meta-models, or between mod-*

els. They aim to specify the links, and their associated semantics, between elements of source and target models" (Jouault et al., 2010). Models are woven by establishing different kinds of links denoting the semantics of weaving: merge operations, traceability links, data translation mappings, text to graphical representation, etc. Atlas Model Weaver (AMW) includes a transformation mechanism with ATL¹ to create an automatic weaving. Virtual EMF (Brunelière and Dupé, 2011) provides a visual assistant to edit two models from different meta-models and to create links between concepts with *drag and drop*. Unfortunately, editors are not supported since the 4.x versions of Eclipse. The existing tools did not suit to our requirements but we got inspired by them to create our own weaving assistant, including improvements and new features (see Section 5.3). Didonet et al. (Del Fabro and Valduriez, 2007) propose an approach that uses matching transformations and weaving models to semi-automate the development of transformations, which is not our goal. In that case, weaving can implement transformations but cannot map models, which is our goal. Our automated mappings implementation has been inspired by this work.

Model mapping has also been explored in the context of Domain Specific Languages (DSL) to define new languages from existing ones in a non-invasive way without re-creating the tool support. Brunelière et al. (Brunelière et al., 2015) define a textual DSL with extension operators to extend meta-model semantics. It is independent from modelling tooling. Similarly, Greifenberg et al. propose DSL-specific tag language (Greifenberg et al., 2016). Kolovos et al. (Kolovos et al., 2010) propose decorator extraction and injection operators based on GMF notes to ensure the non-invasive property but it requires manual transformations and conflicting specializations of GMF notes may appear. Langer et al. (Langer et al., 2012) propose EMF profiles, a lightweight adaptation of UML profiles to extend meta-models with annotations, constraints and stereotypes. These DSL extension approaches have in common to work on the (binary) inheritance relation (one model is more specialised than another) while we target any kind of n-ary relations between models such as aggregation, composition, inheritance, dependency *e.g.* traceability... However they are complementary.

In a previous work (Pepin et al., 2016), we compared different model mapping techniques such as merging, weaving, and annotation between meta-models and we showed that they do not support properly the four above mentioned mandatory criteria for model maintenance and evolution.

¹<http://eclipse.org/atl/>

EMF Facet. In MDA, it's common to use tools based on the Eclipse EMF Frameworks. EMF enables to define meta-models, load, persist and manipulate the compliant models. The EMF Facet tool is based on EMF to extend virtually a meta-model. This solution answers to the non-intrusiveness and the semantics criteria. However the existing EMF Facet tools do not fulfil all requirements: they have no persistence support for the mapping links (the values are computed by queries only) and few navigation and semantics support. However this was the closest approach conforming to our requirements so we decided to improve EMF Facet to draw links between models and manage them in conformance to the defined cardinalities.

2.2 Requirements for Meta-model Extension

We can now revisit the criteria of the beginning of this section to describe the requirements for a *Non-intrusive but Persistent Model Mapping* technique (NIP-MM).

- Mapping meta-models in a unique meta-model usually breaks the evolution lifecycle: when the individual models change, the mapping becomes inconsistent and the associated tools obsolete. We need a technique to map meta-models **without intrusion** and **without version dependency**.
- The weaving and annotation techniques are non-intrusive but they use only generic links while the end-user model transformation tools require specific information to proceed adequate transformations according to the meta-model relations and cardinalities. The mapping technique must include **semantic information** for these relations.
- Last, the technique must **serialize** the mapping links to support persistence in a way that loading model mappings enables the **navigation** between the original model elements and the new one without disturbing the end-user experience. Behind these properties we find the ascending compatibility and the version preservation of existing tool suites which are industrial concerns. Only the Facet approach covered these requirements, except persistence and navigation facilities which are not supported.

In summary, we require an equipped Non-intrusive Model Mapping technique supporting semantic links and persistence.

3 REVISITING THE FACET APPROACH

In this section we overview the EMF Facet features, its interest and limitations and we introduce improvements.

3.1 The Basic EMF Facet

Eclipse EMF Facet is a runtime meta-model extension framework composed of four parts: Facet, Customization, Widgets and Query. The Facet part offers the possibility to virtually extend (at runtime) existing meta-models and models. The Customization part adds UI enhancements on a meta-model. The Widgets part can be used to apply customizations to model editors. The Query part enables to compute attribute, reference and operation value. The queries are written in Java or OCL. With Facet one can **extend models** by adding virtual features to existing models and also **weave models** by linking their concepts (Figure 1). This paper is mainly concerned with the *Facet* part; we dealt with customizations when handling specific meta-models editors in (Pepin et al., 2016).

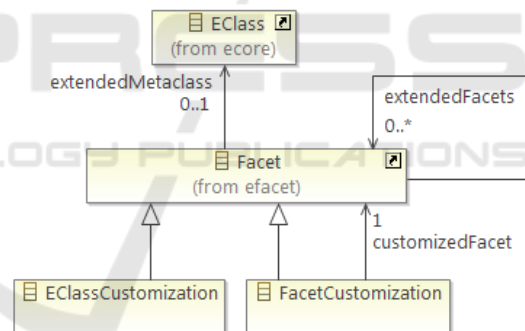


Figure 1: Facet and Customization.

A Facet provides a new viewpoint on a model which is helpful to categorize model elements with new classifications, to add information on model elements, to navigate easily between model elements with new derived links. A facet provides a virtual mechanism to add new attributes, references or operations on a model without modifying the initial meta-model. Several facets can co-exist and be loaded/unloaded on demand without re-opening the model instance. Under the hood, the Facet meta-model extends the meta-class (EClass) from the EMF Ecore meta-model. Facet applicability is checked by optional conformance rules.

As illustrated in Figure 2, the meta-model of a Facet may contain FacetAttribute (extend EAttribute in Ecore), FacetReference (extend EReference in

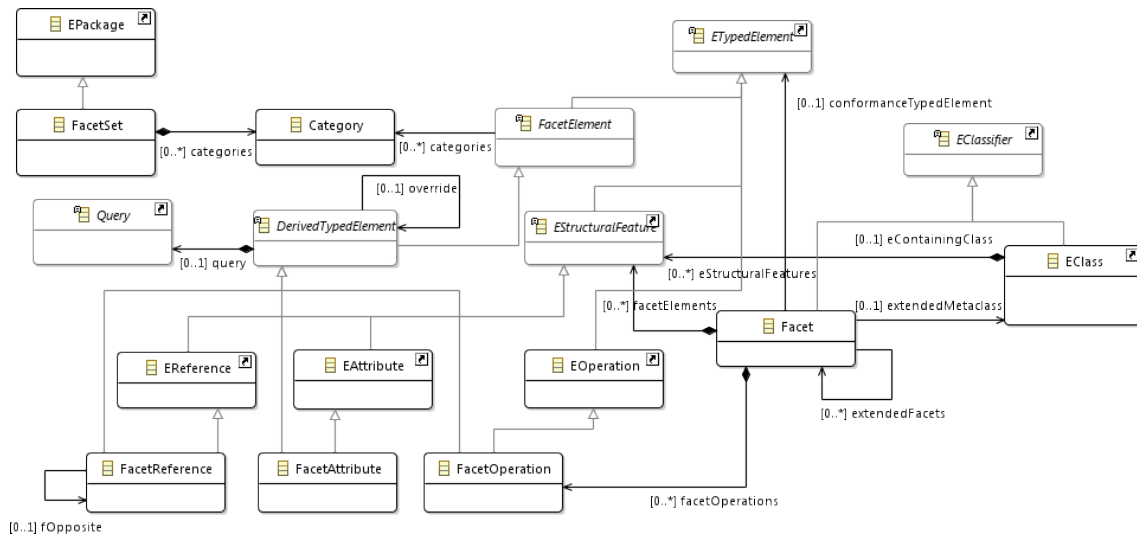


Figure 2: eFacet class Diagram.

Ecore) and FacetOperation (extend EOperation in Ecore) which return values based on query evaluation. Facets are contained in a FacetSet (extend EPackage in Ecore), and FacetSet can be contained in another FacetSet hierarchically.

3.2 Common Use of Facets

Facet is an answer to different use cases in MDA practice. We illustrate with three situations.

- Facet can be used to implement UML *derived features* i.e. attributes or associations that are not implemented but computed from other features (also called daemons in programming). For example, the age is computed from the date of birth. Facet enables one to represent them as attributes or references² without storing redundant informations.
- Facet can be used to customise instances of a meta-model for GUI (user interface) facilities (labels, icons, color...). Facet enables to customize Eclipse SWT components like trees, arrays, lists, etc. As an example, Figure 3 shows a software component classification (components and applications). We modify the GUI by separating components and application types and the associated icons in Figure 4.
- A third case is to add new data in a model; new features (attributes or references) which are not computed but stored as standard features.

These examples improve the practice of model (and meta-model) maintenance and evolution but also model transformation. New information can be added

²References are the way to represent associations at the implementation level.

- ▷ Application Component SAFIG
- ▷ Application Component ASPHERIA
- ▷ Application Component MIKROS
- ▷ Application Component PN_ContratsIARD
- ▷ Application Component 128 Consultation RAQVAM 1ere Generation

Figure 3: Before the GUI modification.

- ▷ Component SAFIG
- ▷ Component ASPHERIA
- ▷ Application MIKROS
- ▷ Application PN_ContratsIARD
- ▷ Application 128 Consultation RAQVAM 1ere Generation

Figure 4: After the GUI modification.

without breaking the source meta-models and without requiring a migration to a new meta-model.

This paper is a core contribution to that problem because the basic Facet does not allow this case. We call it *Non-intrusive but Persistent Model Mapping* technique (NIP-MM).

Limitations. We remind the limitations of the Facet approach for NIP-MM:

- A new feature of the Facet systematically calls a query. However the queries cannot access to the model to map and it is necessary to store the values. Also queries must be executed during the model loading; if the model is voluminous, the computation times can impact the response time.
- New features cannot be valued manually, as the attributes or references of an ordinary meta-model.
- In independent model composition, the mapping links must be persistent, consequently the values of the features is to be serializable.

3.3 Facet Improvements

To fix the above limitations, we first improve the meta-model, then we modify the existing Facet manager and serialization mechanisms.

Facet Manager. We improve the meta-model by allowing dynamic structures instead of the static one. We add *FacetAttribute* and *FacetReference* getters and setters instead of recomputing systematically the attached query (Figure 5). Since the meta-model constraints have an impact on the behaviour of the current Facet manager, we upgraded its implementation to support the new behaviour.

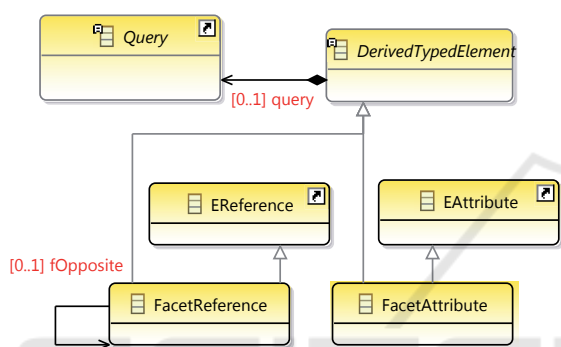


Figure 5: EMF Facet meta-model modifications.

Technically, enabling the access of values turns to weakening the multiplicity of *FacetAttribute* and *FacetReference* which extend *DerivedTypeElement*. Thus the multiplicity of the query *eReference* on *DerivedTypeElement* is changed from 1..1 to 0..1. This feature will permit to get the values of *FacetReference* and *FacetAttribute* without using a query. Furthermore we add a new *eReference* named *fOpposite* to create a reflexive reference mechanism like *eOpposite* on *EReference* in the Ecore meta-model. The improved EMF Facet enables now to manually extend and weave models. As a matter of fact, we fulfil the criterion of Section 2.1: the serialization makes the improved Facet be the most effective approach among all the mapping techniques of Section 2. We submitted our proposal of extended EMF meta-model as a contribution to the open-source Eclipse EMF Facet³; this has been approved.

Serialization Mechanisms. After the modification of the meta-model, we turned to the Java implementation of the new behaviour of the Facet engine called *FacetManager*. At first, we supported the two

³Available since version 1.0: https://bugs.eclipse.org/bugs/show_bug.cgi?id=463898

simplest multiplicities of the bi-directional references type: one-to-one and many-to-many. But, this was insufficient to cover all the cases so we proceeded with all the possible cases. In the next section, we present the cases and their implementation in order to obtain a complete weaving engine.

4 PERSISTENCE AND NAVIGATION

At this stage, one can create *FacetReference* links to map concepts from different meta-models. In this section, we describe the mechanism that supports bi-directional mappings and a two-way navigation.

Mappings are represented here by UML associations, which are bi-directional by default. An association end multiplicity (or cardinality) is an interval of value represented by a lower and an upper bound. An association between classes is represented by a set of links at the instance level.

At the implementation level, maintain a link between elements is an *object relation mapping* (ORM) problem (Ambler, 1997) where the link multiplicities and bi-directional navigation are ensured. This problem needs a robust algorithm engine to maintain the constraints imposed by the multiplicities. A (bi-directional) association is represented by a pair of uni-directional (one-way) associations as shown in the Library example of Figure 6. Thus any (instance)

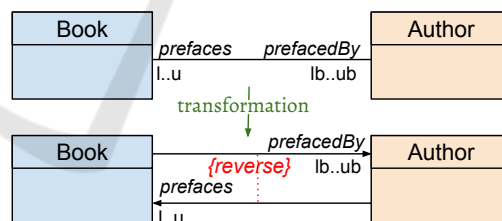


Figure 6: Bi-directional link example.

link modification must be propagated to the opposite link. Depending on the multiplicity values, four cases are distinguished: **one to one** (an instance is linked to one instance only), **one to many** (an instance can be linked to several instances), **many to one** (several instances can be linked to one instance), **many to many** (several instances can be linked to several instances). Note that *one to many* and *many to one* cases are not symmetric because the relation is oriented. Then, these two cases have different algorithms to preserve the multiplicity.

The core issue is to preserve the symmetry constraint (`prefacedBy.reverse() = prefaces`) of the opposite association ends. Updating only one side usu-

ally leads to a symmetry constraint violation. In the following, we illustrate each case by giving a figure and the algorithm we implemented in FacetManager⁴. **One to One.** In Figure 7, a book is prefaced by one author and an author prefaces only one book. To check the multiplicity consistency, the implementation in the FacetManager involves to keep only one opposite link during the set operation.

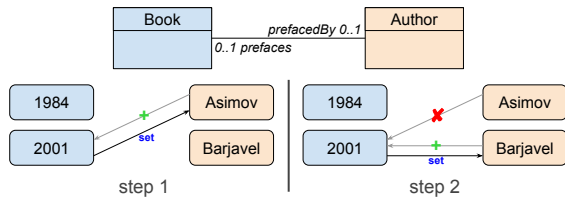


Figure 7: One to one relation.

- 1 delete old reference
- 2 delete old opposite reference
- 3 create new opposite reference
- 4 create new reference

One to Many. In Figure 8, a book is prefaced by many authors and author prefaces one book. To check the multiplicity consistency, the implementation in the FacetManager involves to replace the opposite link during the set operation.

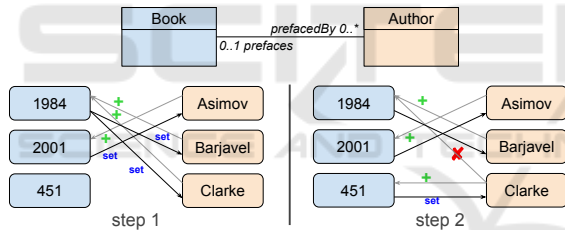


Figure 8: One to many relation.

- 1 remove old reference from existings
- 2 delete old opposite reference
- 3 create new opposite reference
- 4 add new reference into existing

Many to One. In Figure 9, a book is prefaced by one author and author prefaces many books. To check the multiplicity consistency, the implementation in the FacetManager involves to delete the old and to add the new opposite link during the set operation.

- 1 delete old reference
- 2 remove old opposite reference from existings
- 3 add new opposite reference into existing
- 4 create new reference

Many to Many. In Figure 10, a book is prefaced by many authors and author prefaces many books. To check the multiplicity consistency, the implementation in the FacetManager involves to delete the old and to add the new opposite link during the set operation.

⁴Available since version 1.1: https://bugs.eclipse.org/bugs/show_bug.cgi?id=510039

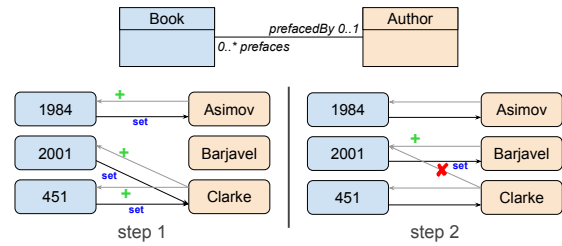


Figure 9: Many to one relation.

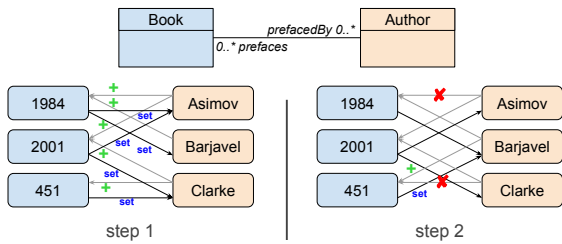


Figure 10: Many to many relation.

- 1 remove old reference from existings
- 2 remove old opposite reference from existings
- 3 add new opposite reference into existing
- 4 add new reference into existing

Thanks to the automatic management of bi-directional links, the instance mapping is transparent for users. Our implementation allows one to weave the instances without a slave-master semantics. The user can draw a link in any order: from the source to the target or reverse.

Testing New Implementations

According to the *Test-Driven Development* approach, *JUnit* tests have been written to check our implementation and its code coverage before adding it to the *FacetManager*. We illustrate this with two simple examples of the *Library*. For each multiplicity case, we set *facetReference* between different instances of books and writers, then we check if the reference setting conforms to the reference and its opposite.

Example 1. In the many-to-one case, *writer1* and *writer2* write the preface of *book1*. The test case of Listing 1 assigns the value, establishes the mapping and checks the reverse link. It succeeds.

Listing 1: Many-to-one: the direct preface reference

```
final FacetReference preface = getFacetRef(
    MANY_TO_ONE, WRITER_EXT, PREFACE);
final Book book1writer1 = this.facetMgr.
    getOrInvoke(writer1, preface, Book.class);
Assert.assertEquals(msg(WRITER1, BOOK1), book1,
    book1writer1);
final Book book1writer2 = this.facetMgr.
    getOrInvoke(writer2, preface, Book.class);
```

```
Assert.assertEquals(msg(WRITER2, BOOK1), book1,
    book1.writer2);
```

Example 2. The test case of Listing 2 checks the automatic setting of the opposite reference prefaced: book1 is prefaced by writer1 and writer2. Its execution also led to success.

```
Listing 2: Many-to-one opposite: the prefaced reference
final FacetReference prefaced = getFacetRef(
    MANY_TO_ONE, BOOK_EXT, PREFACED);
final List<Writer> writers1and2 = this.facetMgr.
    getOrInvokeMultiValued(book1, prefaced, Writer
        .class);
Assert.assertEquals(msg(BOOK1, "writer1_and_writer
    _2"), Arrays.asList(writer1, writer2),
    writers1and2);
```

The above examples are specific to the Library case study but generic tests, written at the meta-model level, can be shared by all applications. Henceforth EMF Facet enables one to create links between any models following a definition at the meta-model level. Section 5 presents additional tools to handle the links at the instance level.

5 END-USER RUNNING THE MAPPING

The mapping API and the tests have been presented in Section 3 and Section 4. In this section, we illustrate the implemented industrial tools associated to the mapping framework with the Library example.

We extend *Book* and *Writer* meta-classes by creating new *FacetSet* for each link mapping cases containing all facets which define the new virtual references with the specific multiplicity: *prefaces* and *prefaced*. A mapping process includes the following activities: *facet mapping definition*, *instance valuation*, *instance linking*, *mapping navigation and evaluation*.

5.1 Mapping Definition

Each facet defines at least a name and the meta-class type. A facet optionally extends an existing Facet. The facet refers to the original meta-class by its absolute *Universal Resource Identifier* (URI). The facet defines three kinds of features: *FacetAttribute*, *FacetReference* and *FacetOperation*. We can create as many new features as required. A *FacetReference* defines at least a name, a multiplicity, a type and an opposite reference if the association is bidirectional. The type is a meta-class from any Ecore reachable

meta-model or another predefined *FacetReference* in the current *FacetSet*. A *FacetAttribute* defines at least a name, a multiplicity and the meta-class type as in *FacetReference*.

Example. Assume a *Book* meta-model describing the books and a *Writer* describing authors. In Figure 11, we create four *FacetSets*, one for each mapping multiplicity: many to many (MToM), many to one (MToO), one to many (OToM), and one to one (OToO). For each case, a new reference *preface* links 'writers to books' and the opposite reference *prefaced* links 'books to writers'. The multiplicity differs through the upper and lower bound. In the example of Figure 11, -1 represents the 'many' upper bound value. The purpose of this new definition is to extend the existing classification of a library of books and authors to obtain an enriched catalogue.

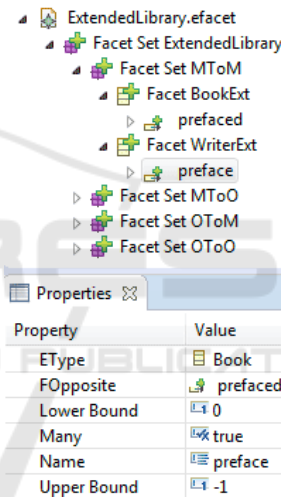


Figure 11: FacetSet definition example.

5.2 Setting Links by Property Value

Property editors set the value of the attributes and references that will be used by queries.

Example. To experiment the different multiplicity links, we create a library with books and writers. We apply a specific *FacetSet* at a time: MToM, MToO, OToM, or OToO. In Figure 12 the *FacetSet* OToM is applied, the book "2001 Space Odyssey" have two prefaced writers "Isaac Asimov" and "Arthur C. Clarke". The property field varies according to the multiplicity: a pop-up menu selector (one) or a double selector (many). We can check that the opposite link is correctly set in Figure 13: "Isaac Asimov" *preface* "2001 Space Odyssey", and "Arthur C. Clarke" *preface* "2001 Space Odyssey".

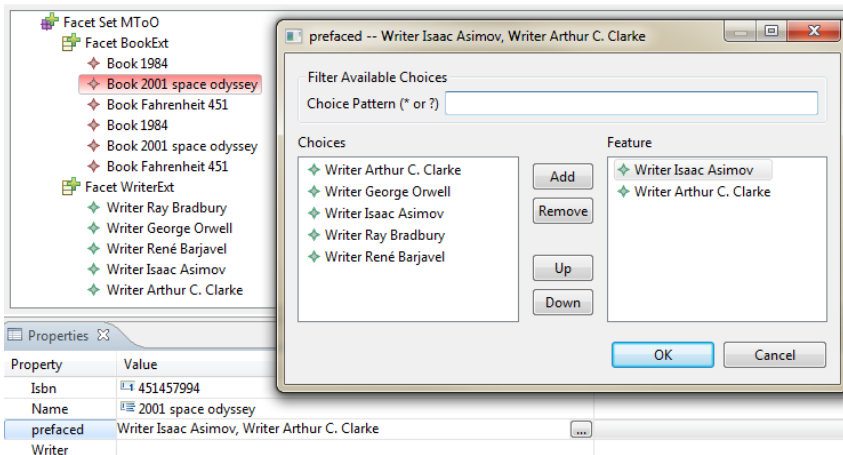


Figure 12: Editing multi-valued reference.

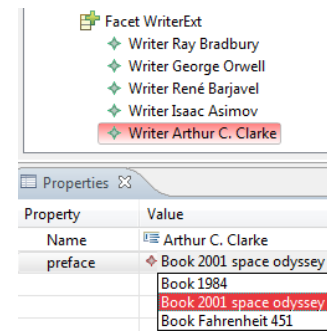


Figure 13: Editing mono-valued references.

5.3 Setting Links by Model Weaver

Setting many links between instances with the property values is quite fastidious and editors are really useless in this case. We developed a specific weaving editor with multiple views (drag-and-drop).

Example. On the right part of Figure 14, an outline displays the different models to weave *i.e.* a first model 'library with books' and a second model 'library with writer'. On the left, a specific view organizes the weaving result by facets. The MToM facet is loaded. This design allows us to drag and drop elements from right to left to link elements by references corresponding to the FacetSet definition. In this example, we drag and drop two writers "Ray Bradbury" and "George Orwell" on reference *prefaced* of the book "Fahrenheit 451".

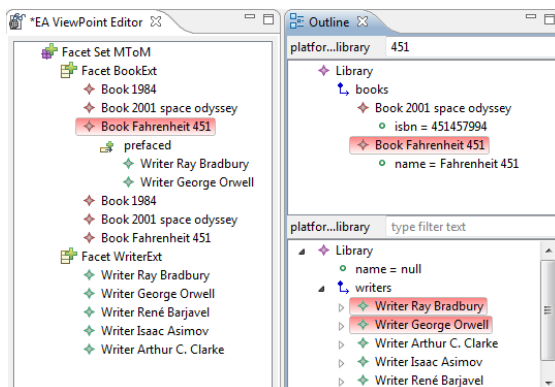


Figure 14: Model weaver editor.

5.4 Exploring Links with OCL Query

Using a model mapping by FacetReferences makes possible the navigation through the models, from one

FacetReference to another. The *TreeEditor* is compatible with EMF Facet and enables one to browse hierarchically the models by the FacetReference, but it is still not very convenient; architects would like to get some statistics to measure which elements from models are mapped or not. Since EMF offers OCL expressions parser on Ecore models, we have produced an engine extension in order to make Facet compatible with virtual features and used as property in OCL statements. This extension is also integrated in a plugin to EMF Facet project.

Example. In Figure 15, we use the previous model weaving result. We open the model 'library with books' and write an OCL query on the console to know all writers who have prefaced the books:

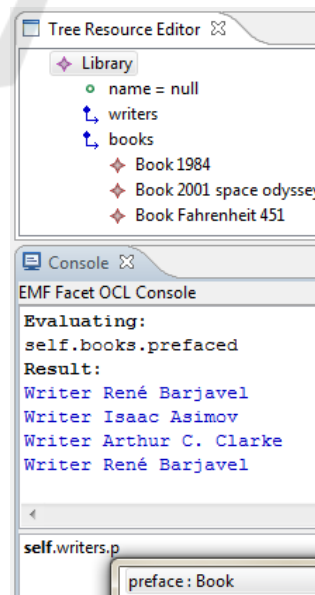


Figure 15: OCL Facet Console.

"*self.books.prefaced*". The query writing is assisted with auto-completion, and the result appears on the console.

Section 6 draws experimentations of the above tools and wizards.

6 EXPERIMENTATIONS

The above tools and wizards (and others) have been integrated in the EMF Facet open-source project. The improved Facet tooling is available at the open-source Eclipse EMF Facet https://wiki.eclipse.org/EMF_Facet.

6.1 Starter Kit Example

EMF Facet remains a framework with components to be integrated by developers in their applications. This Eclipse project has no introductory examples to show how to integrate the components to make an application, to handle models compatible with facet extensibility or to create a facet set (or facet customization) to extend an existing meta-model.

To illustrate the usability and availability of the contribution of this paper, we pushed some source codes and model examples in a public and open source *Starter Kit* project at: https://gitlab.com/jpepin/EmfFacet_StarterKit/.

Eclipse users can build the *Update Site* to install easily the example in Eclipse installation (compatible with the Kepler and more recent versions).

1. In Eclipse 'Help > Install New Software', enter the update site url https://gitlab.com/jpepin/EmfFacet_StarterKit/raw/master/in.jpep.emf.facet.starterkit.update.site/.
2. As illustrated by Figure 17, select *EMF Facet Starter Kit Feature* and next...

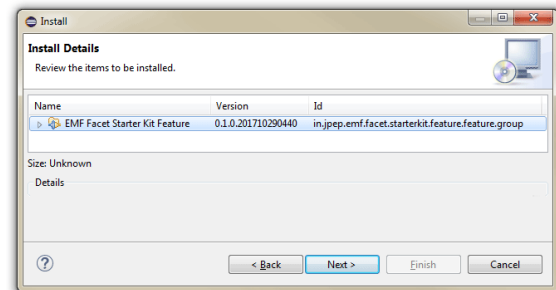


Figure 17: EMF Facet Starter Kit.

3. After the plugin installation, create the example project with 'File > New > Example' and
4. choose *Models Example* from the *EMF Facet Starter Kit* category.

The tiny example is given in Figure 16. Two models are proposed: *human* and *city*.

Three testing scenarios are proposed, as follows.

Intra Model Facet Extension. You can open and compare the files *example.human* and *example.mset*. This *mset* is opened with an editor compatible with Facet and the facet definitions are enabled at the beginning. You can see through the *Properties* view, that the facet extension adds a new attribute 'Surname' and a new bidirectional reference 'parent' / 'children' of *John*. The values of the two new features can be assigned and stored

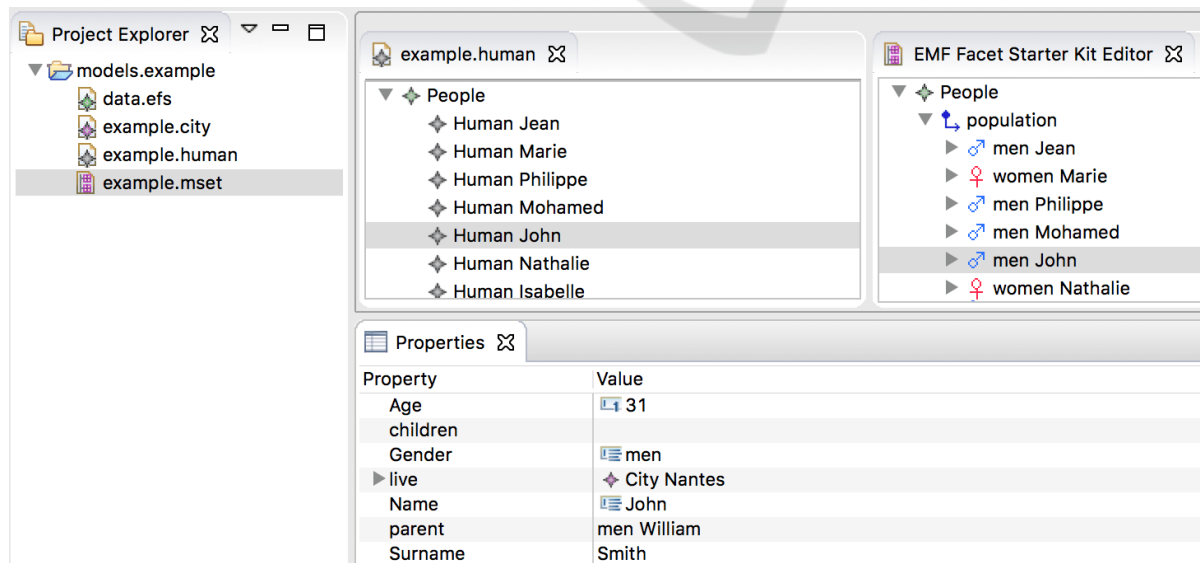



Figure 16: EMF Facet starter kit models example.

in the data file.

Inter Models Facet Extension. The *human* and *city* models are clearly independent. This example shows a new facet extension to add bi-directional reference 'live' / 'habitants' between the models. Similarly to the 'intra' scenario, the values can be updated with the *Properties* view, and saved with result serialized in *data.efs*.

Customization. By default the *human* meta-model does not distinguish the gender man and woman. To add the visual distinction of Figure 16, open the file *example.mset*, press the button 'Load / Unload Customizations' from the tool bar  and choose *Family Custo* definition. Now, men and women are displayed with different prefix label and icon.

If you are software developer, the sources of the example can be download from the *EMF Facet Starter Kit* Git repository. It demonstrates how to implement an editor with facet compatibility integration: extended property view, load / unload facet and customization buttons, set / load facet serialization file. Facet and customization definitions for meta-models Human and City are available too for example and can be widen. Two examples of query are in the customization definition: one in Java, the other in OCL.

This tiny example shows that the facet persistence and navigation contribution is already available in the Eclipse open source project for developers to create simple tools without complex configuration for end-user. Moreover, the example demonstrates that EMF Facet is adapted to different use cases: to add attributes and association in a model; to link two different and independent models; or to customize the user interface. This example can be adapted and extended for largest and real use cases.

6.2 User Stories

We illustrate here two situations of software maintenance and reverse-engineering where the non-intrusive but persistent model mapping is pertinent.

In legacy modernization, the architects need to establish the *as-is* state of the information systems (from business processes to deployed applications). We need to enrich the models with additional information without modifying the meta-models just like source code annotations: deprecated, renaming... These information are useful to develop the *to-be* state of the next generation information systems but must not interfere with the existing programs. Conversely to code annotation, the improved Facets enables to enrich with more than one layer (one layer by aspect we are interested in).

When standard meta-models (UML, BPMN...) do not fit the requirements of one organisation, one can customize the meta-models and create a family of extensions that is consistent with the standard releases, as far as a new release of one standard did not impact too many concepts (like the transition from UML 1.x to UML 2.x). The maintenance of the customizations follow their own life cycles.

6.3 Large Scale Experimentations

The *Library* example was a toy example for illustrating this paper. We experimented our improved Facet in the context of Business-IT alignment in Enterprise Architecture. This context covers a wider field than the scope of this paper. We briefly report here this experimentation but the reader will find more details in (Pepin et al., 2016).

The general problem was to align models from different points of view, in the context of Information System maintenance. We defined different meta-models (business process, functional, application) and we implemented techniques to feed the corresponding models from legacy information (source code, data and models when they exist).

The mapping support, as defined in this paper, was the core technique to establish the alignment links between the models. We experimented real size case studies provided by insurance companies with heterogeneous information supports: lots of java source files, MEGA repositories, databases,... The experiments showed that big mappings are hardly manageable by humans and tool assistance is mandatory. Our tool support handled efficiently big models. Even in bulky case with manual mapping, our Weaving Editor (*cf.* figure 14) provides an optimized user interface with search engine to find concepts and highlight concepts matching. However additional tool is needed to visualize big mappings, to evaluate the mapping properties (consistency, completeness) and quality (misalignment, evolution). Our Facet query engine is a first step to reach this goal.

7 CONCLUSION

We proposed an improved technique of virtual extension of meta-models that uses Facets. It enables one to modify meta-models already in use by software, without rebuilding completely the legacy tool support. The extension takes account of the multiplicity of associations and considers two-way references between the involved entities.

- *A mapping model* We experimented existing techniques for model composition and we propose an improved version of virtual extension of meta-models that uses Facets and enables one to modify meta-models already in use without rebuilding the software product. The extension is called virtual because it does not directly impact the initial meta-models.
- *A mapping technique* which is compliant with the existing MDA tools. EMF enables to define meta-models, load, persist and manipulate the compliant models. The EMF Facet tool is based on EMF to extend virtually a meta-model. This solution is non intrusive, it supports link semantics but it does not have a mechanism for persistence (the values are calculated by queries) and an adequate technique and tools for mapping models. We have implemented this technique which is now integrated to the open-source Eclipse EMF Facet project.
- *A mapping tool* We overcome this limitation by improving the EMF Facet technique, by modifying the meta-model and implementing several tools to manage model mapping in practice. Our implementation preserves the link multiplicities and a bi-directional navigation.

The proposed technique has been implemented, then experimented on various case studies and integrated in the open-source Eclipse EMF Facet project. We have then contributed to solve an important model and software evolution issue.

The next step will provide more assistance to the user; we target the implementation of heuristics to propose a list of possible model mappings to the modeller: he can then choose the desired ones. These heuristics will depend on the nature and the semantics of the mappings. For example, when mapping two releases of the same model, it is usually easier to detect equality mapping. In specific cases, one can detect patterns or naming conventions.

REFERENCES

- Ambler, S. W. (1997). *Building Object Applications That Work: Your Step-by-Step Handbook for Developing Robust Systems with Object Technology*. Cambridge University Press.
- Atlee, J. M., France, R., Georg, G., Moreira, A., Rumpe, B., and Zschaler, S. (2007). Modeling in software engineering. In *Companion to the Proceedings of the 29th International Conference on Software Engineering, ICSE COMPANION '07*, pages 113–114, Washington, DC, USA. IEEE Computer Society.
- Brambilla, M., Cabot, J., and Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers.
- Brunelière, H., García, J., Desfray, P., Khelladi, D. E., Hebig, R., Bendraou, R., and Cabot, J. (2015). On lightweight metamodel extension to support modeling tools agility. In *Modelling Foundations and Applications - 11th European Conference, ECMFA*.
- Brunelière, H. and Dupé, G. (2011). Virtual EMF - transparent composition, weaving and linking of models. In *EclipseCon Europe 2011*.
- Clark, T., Barn, B. S., and Oussena, S. (2011). Leap: A precise lightweight framework for enterprise architecture. In *Proceedings of the 4th India Software Engineering Conference, ISEC '11*, pages 85–94, New York, NY, USA. ACM.
- Clavreul, M. (2011). *Model and Metamodel Composition: Separation of Mapping and Interpretation for Unifying Existing Model Composition Techniques*. PhD thesis, Université Rennes 1.
- Cuadrado, J. S., Izquierdo, J. L. C., and Molina, J. G. (2014). Applying model-driven engineering in small software enterprises. *Sci. Comput. Program*.
- Del Fabro, M. D. and Valduriez, P. (2007). Semi-automatic model integration using matching transformations and weaving models. In *Proceedings of the 2007 ACM symposium on Applied computing*.
- El Kouhen, A. (2016). Panorama : A Unified Framework for Model Composition. In *15th International Conference on Modularity*, malaga, Spain. MODULARITY 2016.
- France, R. and Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering, FOSE '07*.
- Greifenberg, T., Look, M., Roidl, S., and Rumpe, B. (2016). Engineering tagging languages for dsls. *CoRR*.
- Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., and Bezivin, J. (2010). Inter-DSL Coordination Support by Combining Megamodeling and Model Weaving. In *Proceedings of the SAC 2010*.
- Kolovos, D. S., Rose, L. M., Drivalos Matragkas, N., Paige, R. F., Polack, F. A. C., and Fernandes, K. J. (2010). Constructing and navigating non-invasive model decorations. In Tratt, L. and Gogolla, M., editors, *Theory and Practice of Model Transformations: Third International Conference, ICMT 2010, Malaga, Spain, June 28-July 2, 2010. Proceedings*.
- Langer, P., Wieland, K., Wimmer, M., and Cabot, J. (2012). EMF profiles: A lightweight extension approach for EMF models. *Journal of Object Technology*.
- Marchand, J. Y., Combemale, B., and Baudry, B. (2012). A categorical model of model merging and weaving. In *Proceedings of the 4th International Workshop on Modeling in Software Engineering, MiSE '12*.
- Paige, R. F., Matragkas, N., and Rose, L. M. (2016). Evolving models in model-driven engineering: State-of-the-art and future challenges. *Journal of Systems and Software*.
- Pepin, J., André, P., Attiogbé, C., and Breton, E. (2016). Using ontologies for enterprise architecture integration and analysis. *CSIMQ*, 9.