# Virtual Extension of Meta-models with Facet Tools

Jonathan Pepin[1,2], Pascal André[1], Christian Attiogbé[1] and Erwan Breton[2]
[1]*AeLoS Team LS2N CNRS UMR 6004 - University of Nantes, France*
[2]*Mia-Software - Nantes 44324 Nantes cedex 3*
*{firstname.lastname}@univ-nantes.fr,ebreton@sodifrance.fr*

Keywords: Meta-model; Weaving; Mapping; Facet; Transformation Tools

Abstract: 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 face costly maintenance operations due to frequent new standards and upgraded releases. Therefore, solutions ensuring a better adaptability and flexibility of modelling 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). 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 model and meta-models evolve separately.

This is even more complex when models (including metamodels) are parts of larger models. As mentioned by El Kouhen in (El Kouhen, 2016), MDE promotes the separation of concerns to deal with the design's complexity and maintainability. This practice implies the creation of several heterogeneous models using different notations. Since the semantics of each individual model is limited, the model's consistency and completeness are easier to prove. *Model composition* is the symmetric paradigm of separation of concerns. 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 the context 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), this is a non-intrusive model composition. A *typical scenario* for a software provider is to deliver customised release to some clients but customisation leads to a tricky maintenance problem when models and meta-models evolve. This includes the case of (old) legacy code built using various DSL with various development methods at different level of abstract (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 requiring tool adaptation.

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.

**A mapping model** 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** Taking this 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 (e.g. tools based on the Eclipse EMF Frameworks) and efficient for large-scale applications. 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.

**A mapping tool** We contribute here to this limitation by improving the EMF Facet technique by modifying the meta-model and implement several tools to manage model mapping 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 multiplicities 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 while larger case studies are reported in Section 6. Finally, Section 7 summarises the contribution and draws perspectives.

## 2 Background and Requirements

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. Model composition is usually applied at the meta-model level.

Non-intrusive Model Mapping (NI-MM for short), a composition that preserves the components, is called the "*model-based correspondence*" by Clavreul who identified 88 techniques for different purposes in his systematic review of models composition (Clavreul, 2011). 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. A mapping tool is proposed by Limyr et al. (Limyr et al., 2006) but the goal is to prepare patterns for model transformation, not to map heterogeneous models. 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. NI-MM can be seen as an *ad hoc* model composition in their classification since the mapping uses semantic information of the source models. Their mapping approaches enable the separation of concerns, but the merging and weaving do not preserve the legacy models but also tool support because their result is a new model.

NIP-MM 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 models. 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 that contains 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 ATL[1] 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 no longer updated and no more supported for the recent Eclipse 4.x versions. The existing tools did not

[1] `http://eclipse.org/atl/`

suit to our requirements but we got inspired by them to create our own weaving assistant, including some improvements and new features (see Section 5). 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.

NI-MM 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. Bruneliere et al. (Brunelière et al., 2015) define a textual DSL with extension operators to extend metamodel 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 requires manual transformations. Also 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. An advantage is to add new information. 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.

As mentioned in a previous work (Pepin et al., 2016), mapping approaches such as merging, weaving or annotation do not support properly mandatory properties for model maintenance and evolution. These properties issue from lessons learned during model maintenance activities. A software system sustains several releases that can even co-exist for different users with different hardware. The semantics of models needs enhancements: new attributes, new entities, new classification, and new links....

- Mapping meta-models in a unique meta-model usually breaks the evolution lifecycle: when the individual models (or meta-models) change, the mapping become inconsistent and the associated tools are obsolete. Model merging or extension are such intrusive techniques since the initial model disappears in the resulting model. 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 user-end 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 this requirements, except persistence and navigation facilities are not supported.

In summary, we require an equipped NI-MM technique supporting semantic links and persistence.

## 3 Revisiting the Facet Approach

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 your own customization to model editors. The Query part enables to compute attribute, reference and operation value. The query can be 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).

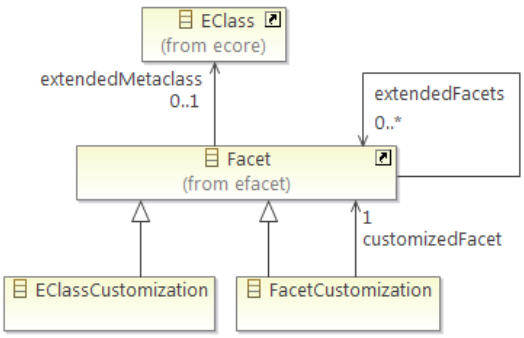


Figure 1: Facet and Customization

A Facet provides a new viewpoint on a model which is helpful to categorize model elements with new classification, to add information on model element, to navigate between model elements easily with new derived links. A Facet provides a virtual mechanism to add more 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. Facet applicability is checked by optional conformance rules.

Let us remind here 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 calculation 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.

To fix the above limitations, we first improve the meta-model, then we modify the existing Facet manager and serialization mechanisms. We improve the meta-model by allowing dynamic structure instead of the static one: we add *FacetAttribute* and *FacetReference* getters and setters instead of recomputing systematically the attached query (Figure 2). 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.

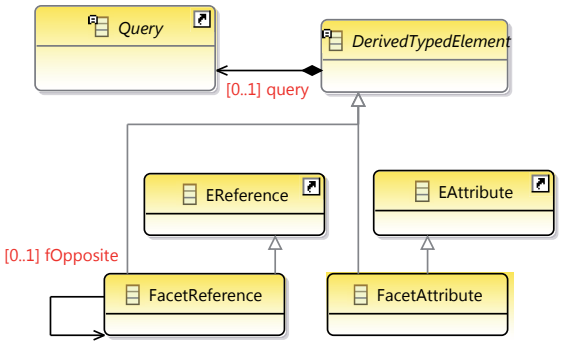


Figure 2: EMF Facet meta-model modifications

Technically, enabling the access of values turns to weakening the multiplicity of *FacetAttribute* and *FacetReference* which extend *DerivedTypeElement*. Thus the multiplicity of the *eReference* query on *DerivedTypedElement* 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, the serialization makes the improved Facet be the most effective approach among all the mapping techniques of Section 2. Consequently we submitted it as a contribution to the open-source Eclipse EMF Facet[2] which has been approved.

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

We can now 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 is bi-directional by default.

Similarly to *object relation mapping* (ORM) or UML2Code transformations, a (bi-directional) association is represented by a pair of uni-directional (one-way) associations as shown in the `Library` example of Figure 3. Thus any (instance) link modification requires a propagation on the opposite link. An association between classes is represented by a set of links at the instance level.

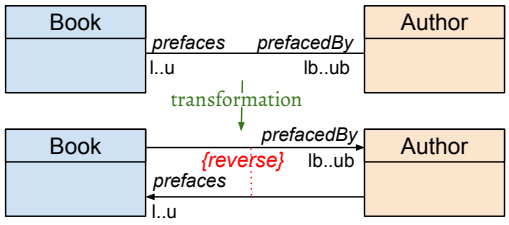


Figure 3: Bi-directionnal link example

We distinguish four cases:

- **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.

[2] `https://bugs.eclipse.org/bugs/show_bug.cgi?id=463898`

Multiplicity is an interval of value represented by a lower and an upper bound. We precise that *one to many* and *many to one* cases are not symmetric because the relation is directional. Then, these two cases have different algorithms to preserve the multiplicity. The problem is to preserve the symmetry constraint (`prefacedBy.reverse() = prefaces`) of the opposite association ends. Updating only one side usually leads to a symmetry constraint violation. Next we illustrate each case by giving a figure and the algorithm we implemented in FacetManager[3].

**One to one**. In Fig. 4, a book is prefaced by one author and an author prefaces only one book. To check the multiplicity consistency, the implementation in the FacetManager involve to keep only one opposite link during the *set* operation.

```
1  @delete old reference@
2  &delete old opposite reference&
3  &create new opposite reference&
4  @create new reference@
```

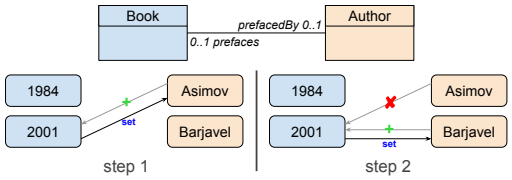


Figure 4: One to one relation

**One to many**. In Fig. 5, 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.

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

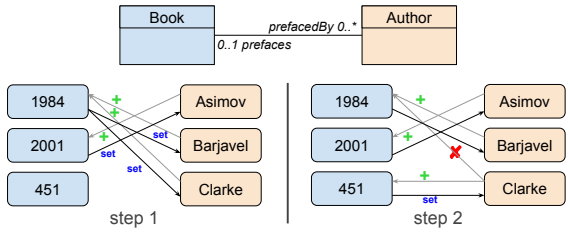


Figure 5: One to many relation

**Many to one**. In Fig. 6, 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@
```

[3] https://bugs.eclipse.org/bugs/show_bug.cgi?id=510039

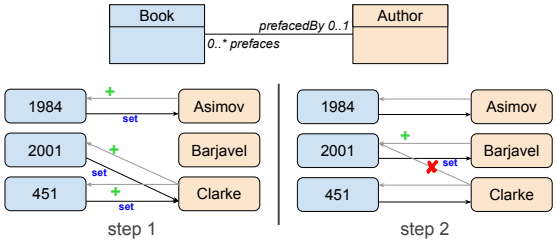


Figure 6: Many to one relation

**Many to many**. In Fig. 7, 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.

```
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@
```

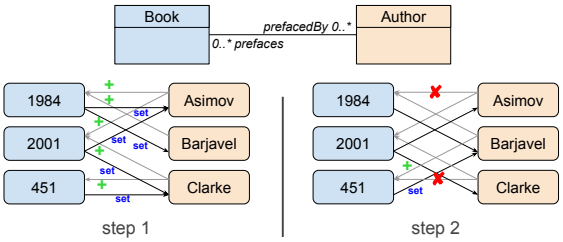


Figure 7: Many to many relation

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. Henceforth EMF Facet enables one to create links between any models following a definition at meta-model level. Section 5 presents additional tools to handle the instance level links.

## Checking Link Conformity by Testing

To test and verify code coverage our new implementation for the mapping added in the *FacetManager*, we wrote *JUnit* tests following the *Test-Driven Development* approach. Let's illustrate it 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 is conform 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,
    book1writer2);
```

**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.

## 5 End-User Running the Mapping

The mapping framework includes an API, a user interface and tests. The mapping API and the tests have been presented in section 3 and 4. In this section, we illustrate the additional implemented tools with the `Library` example. We extend *Book* and *Writer* meta-classes by creating new *FacetSet* for each link mapping cases containing all facets which defines 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*.

### 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.

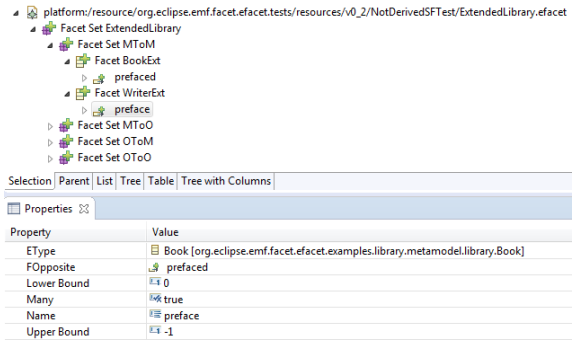


Figure 8: FacetSet definition example

**Example** Assume a *Book* meta-model describing the books and a *Writer* describing authors. In Fig. 8, 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 Fig. 8, 'many' has 0 lower bound and -1 upper bound.

The purpose of this new definition is to extend the existing classification of a library of books and authors to obtain an enriched cataloguing.

### Setting links with property value

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

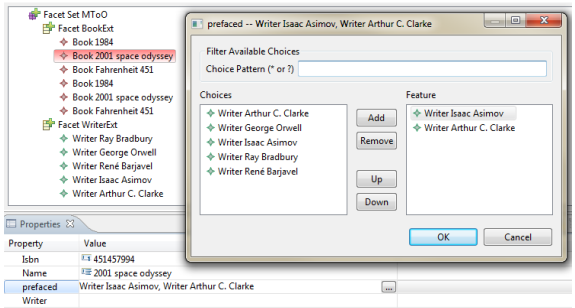


Figure 9: Editing multi-valued reference

**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 Fig. 9 the FacetSet OToM is applied, the book "2001 Space Odyssey" have two prefaced writers "Isaac Asimov" and "Arthur C. Clarke".

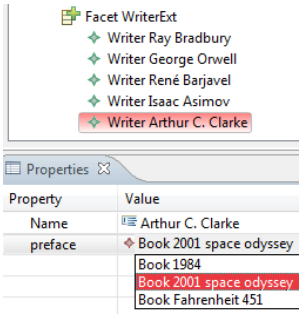


Figure 10: Editing mono-valued references

The property field is different according to the multiplicity: one, a pop-up menu selector ; many, a two columns chooser. We can check the opposite link is correctly set in Fig. 10: *"Isaac Asimov" preface "2001 Space Odyssey"*, and *"Arthur C. Clarke" preface "2001 Space Odyssey"*.

### Setting links with model weaver

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

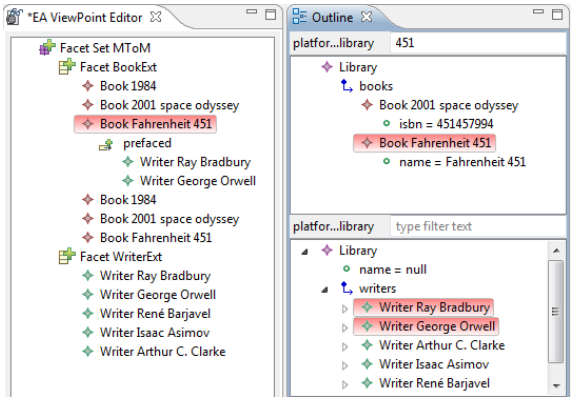


Figure 11: Model weaver editor

**Example** In the right part of Fig. 11, 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*".

### 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 virtual feature compatible and used as property in OCL statements. This extension is also integrated in a plug-in to EMF Facet project.

**Example** In Fig. 12, 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: "*self.books.prefaced*". The query writing is helped with auto-complete, and result appear on the console.

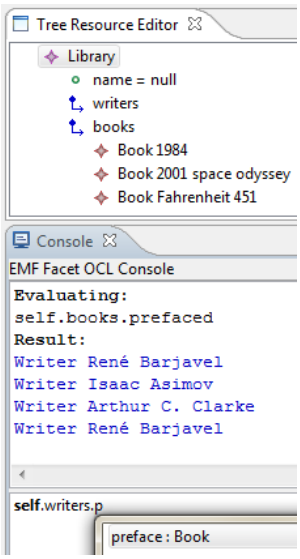


Figure 12: OCL Facet Console

Most presented tools and wizards have been integrated in the EMF Facet open-source project. They are free available for installation on Eclipse IDE.

## 6 Large Scale Experimentations

The `Library` example is a toy example for illustrating this paper. We applied our improved Facet in the context of the Business-IT alignment in Enterprise Architecture. The problem was to align models from different points of view, in the context of Information System maintenance. It covers a wider field than the scope of this paper but we will focus only on the mapping issue. We briefly report these experimentations here but the reader will find details in (Pepin et al., 2016).

In this context, we experimented our Facet approach with real size case studies provided by insurance companies with heterogeneous information supports: lots java source files, MEGA repositories, databases,... First, we defined different meta-models (business process, functional, application) and we implemented reverse-engineering techniques to feed the

corresponding models. The mapping technique has been implemented to establish the alignment links between the models. 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 11) 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.

#### Applications

The improved Facet tooling is available at the open-source Eclipse EMF Facet `https://wiki.eclipse.org/EMF_Facet`.

Application use cases include implementation of heterogeneous modelling (*e.g.* static vs dynamic aspects), traceability links, evolution links (between releases), refinement, roundtrip representation (*e.g.* code vs models), ...

## 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. 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.

Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-Driven Software Engineering in Practice.

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.

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.

Langer, P., Wieland, K., Wimmer, M., and Cabot, J. (2012). EMF profiles: A lightweight extension approach for EMF models. Journal of Object Technology.

Limyr, A., Neple, T., Berre, A.-J., and Elvesæter, B. (2006). Semaphore – a model-based semantic mapping framework. In Proceedings of BPM'06.

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.