A Practical Approach to Model Extension

Mikaël Barbero[1], Frédéric Jouault[1,2], Jeff Gray[2], and Jean Bézivin[1]

[1] ATLAS Group, INRIA and LINA,
University of Nantes, France
`Firstname.Lastname@univ-nantes.fr`
[2] Department of Computer and Information Sciences,
University of Alabama at Birmingham
`Lastname@cis.uab.edu`

Abstract. In object technology, reusability is achieved primarily through class inheritance. In model engineering, where reusability is also important, it should be possible to extend a modeling artifact in a similar manner to add new capabilities. This paper presents a conceptual and practical approach to model extensibility, in which new models are created as derivations from base models. There are several situations where such an extensibility mechanism is useful and essential (e.g., in the case of hierarchies of metamodels). In order to achieve the goal of model extension, a precise definition of the extension mechanism is needed, based on a strict model definition. After describing the context of model extension through a motivating example, the paper outlines a practical implementation with characterization of its main conceptual properties. The solution is being implemented as part of the AMMA model engineering platform under Eclipse.

1 Introduction

Model-Driven Engineering (MDE) offers an advantage due to its power in providing a homogeneous view of heterogeneous artifacts [1]. The main assumption leading to this power is summarized by "everything is a model"; i.e., models are considered as a unifying concept in software.

The work presented here is based on precise definitions of the principles of MDE given in [5] and [7]. According to these previous works, MDE relies on two main relations: *conformance* and *representation*. In this paper, we introduce a third relation called *extension*.

The *conformance* relation links one model to another model called its *reference model*. Throughout this paper, we abbreviate the conformance relation as *c2* (for "conforms to"). Figure 1 illustrates this definition.

Although this first definition allows an indefinite number of conformance layers, the layers must stop at some level for practical purposes. This is accomplished by giving the definitions of the three different kinds of models encountered in the OMG modeling stack:

1. A *metametamodel* is a model that is its own reference model (i.e., it conforms to itself),
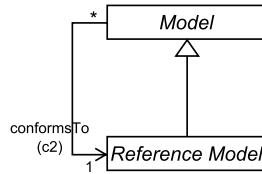
Fig. 1. Definition of a model and its reference model

2. A *metamodel* is a model such that its reference model is a metametamodel,
3. A *terminal model* is a model such that its reference model is a metamodel.

These definitions define a modeling architecture based on three levels, which is compatible with the OMG view.

The second foundational relation of MDE is called *representation*, which links terminal models to the systems they represent. This relation is abbreviated *repOf* (for "representation of") and satisfies the principle of substitutability[1] [8]. This relation is illustrated by Fig. 2. The conformance of a terminal model to its metamodel is also depicted.
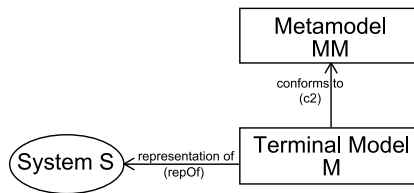

Fig. 2. Basic relations of *representation* and *conformance*

This paper is based on the existence of an additional relation between models called *extensionOf*. Let M_i be a core model representing most concepts for a kind of system. The extension model M_f is a model, defining some new concepts not in M_i but making references to existing M_i concepts. Figure 3 illustrates this relation between the two models M_i and M_f.

The composition of M_i with its extension model M_f leads to a new model M_r (see Fig. 4). M_r is the result model of the composition of initial model M_i and fragment model M_f. This composition does not need to be actually computed, it can be interpreted (i.e., queries over M_r are dynamically translated into queries over M_i and M_f). However, considering the model M_f as an extension of model M_i leads toward a consideration of model M_r.

This paper is organized as follows. Section 2 provides a motivating example. Section 3 describes a corresponding implementation in KM3 [5], and Section 4

---

[1] A model M is said to be a representation of a system S for a given set of questions Q if for each question of this set Q, the model M will provide the same answer that the system S would have provided in answering the same question.
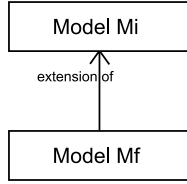
**Fig. 3.** Relation of extension between two models
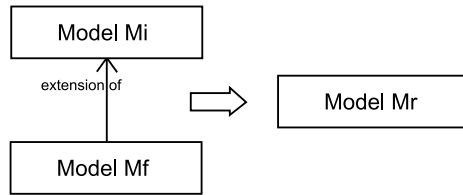
Fig. 4. Extension model and fragment model

gives the conceptual definitions of the model extension mechanism. Section 5 further characterizes this mechanism. Section 6 gives an overview of related works. Section 7 concludes and summarizes future work.

2   Motivating Example

Petri nets are a well-known formalism used to study communication between parallel systems [12]. A classical Petri net is a set of places and transitions linked by directed arcs. Arcs run from a place to a transition or from a transition to a place. The following Petri net (Fig. 5) has four places ($P1$, $P2$, $P3$, and $P4$) and two transitions ($T1$, and $T2$).
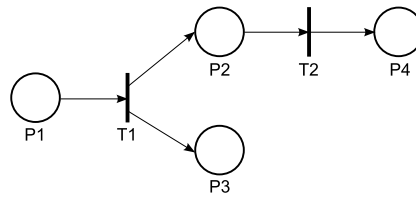
Fig. 5. A simple PetriNet

The metamodel depicted on Fig. 6 specifies all the concepts of a simple Petri nets like the one on Fig. 5. This metamodel describes a Petri net as a set of *Arc*s and *Node*s. A *Node* can be either a *Place* or a *Transition*.
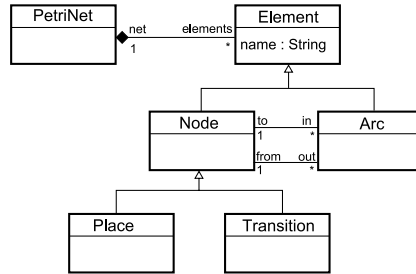
Fig. 6. PetriNet metamodel

From the Petri net of Fig. 5, an infinite number of executions can be launched. The metamodel does not allow the design of Petri nets with a specific execution state. To overcome this limitation, we can *extend* the previous metamodel with the concepts describing the state of a Petri net at a given time. Such a Petri net is said to be marked. A marked Petri net can be represented by attaching a set of *Token*s to some *Place*s. This addition does not affect the previously defined structure of the Petri net.

An initial simplified solution would add an integer attribute to class *Place*. This would give an initial marking by storing the number of tokens associated to a place. However, this would prevent more advanced representations like colored and value-based tokens. Therefore, our application of model extension enables the separate modeling of tokens. Figure 7 shows the same Petri net of Fig. 5 with tokens on places $P1$ and $P2$.
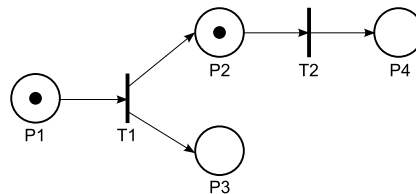

Fig. 7. A simple PetriNet at a given execution state

As we previously mentioned, the base structure of the Petri net with marking is not changed. The metamodel of the Petri net with marking is depicted on Fig. 8. It adds the concept of *Marking* as a set of *Token*s. Each *Token* is associated with a *Place*. The *Place* class comes from the first Petri net metamodel defined in Fig. 6.

This new metamodel is an extension of the original Petri net metamodel. The Petri net of Fig. 7 conforms to this extension. Actually, this Petri net is conforming to the combination of the initial metamodel with its extension, which is the metamodel given in Fig. 9. This combination merges the common concept of *Place* to build the complete metamodel.
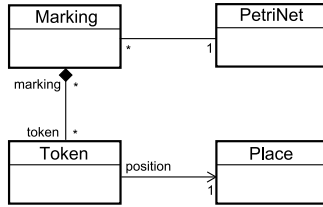
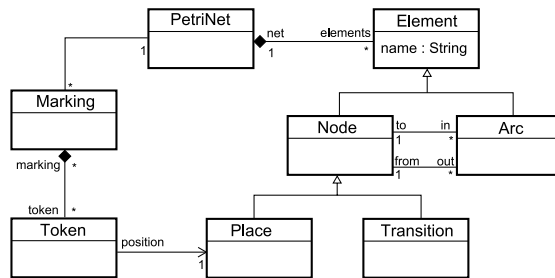**Fig. 8.** Marking metamodel extension of PetriNet

Fig. 9. Result of PetriNet metamodel and marking extension combination

There are many extensions to Petri nets (e.g., colored Petri nets, hierarchical Petri nets, timed Petri nets). The previous mechanism can be used to describe the metamodels of those extensions without having to start anew each time, but by extending the same base metamodel.

3   Implementation Support

In the previous section, an extension to a Petri net metamodel was introduced as a motivating example. This section provides the definition of those metamodels in KM3 [5] format. This format has been the foundation of a conceptual framework that will be described in the next section and extended to a formal definition of model extension.

The initial metamodel of PetriNet describes concepts of *Arc*s, *Place*s and *Transition*s. The metamodel given in Fig. 10 is exactly the same as the one in Fig. 6, but represented using a different (textual) notation. Each concept is a named *Element* having a reference to its owning Petri net. *Place*s and *Transition*s have some input and output *Arc*s. *Arc*s are linked to one source *Node*, and to one target *Node*.

The previous metamodel describes the concepts that are shared between all kinds of Petri nets. An extension of this metamodel has been defined to represent marked Petri nets and is given in Fig. 11. It is the same metamodel as the one presented in Fig. 8. All classes having the same name as one of the classes of

```
1 package PetriNet {
2   class PetriNet {
3     reference elements[1-*] container : Element oppositeOf net;
4   }
5   abstract class Element {
6     attribute name : String;
7     reference net : PetriNet oppositeOf elements;
8   }
9   abstract class Node extends Element {
10    reference in[*] : Arc oppositeOf to;
11    reference out[*] : Arc oppositeOf from;
12  }
13  class Arc extends Element {
14    reference from : Node oppositeOf out;
15    reference to : Node oppositeOf in;
16  }
17  class Place extends Node {}
18  class Transition extends Node {}
19 }
```

Fig. 10. PetriNet metamodel in KM3

the initial metamodel actually correspond to the same classes. Previous features
are not re-defined. For instance, the *PetriNet* class is present in both the initial
metamodel and in the extension metamodel. Only the *markings* reference to
Marking elements is added. The extension metamodel is also adding the concepts
of *Marking* and *Token*. A *Marking* contains a set of *Token*s and a token is
attached to a *Place.*

```
1 package PetriNet {
2   class PetriNet {
3     reference markings[*] : Marking;
4   }
5   class Place {}
6   class Marking {
7     reference petrinet : PetriNet;
8     reference tokens[*] container : Token;
9   }
10  class Token {
11    reference position : Place;
12  }
13 }
```

Fig. 11. Marking metamodel extension of PetriNet in KM3

When the previous metamodel is used as an extension of the initial one, it
can be described as the "composition" of those two metamodels into a single
one. This "composed" metamodel is given in Fig 12. Each concept defined in
one of the two previous metamodels is present in this new metamodel. When a
concept exists in both previous metamodels (like *Place* and *PetriNet*), the result
of the extension is the merging of elements from the initial metamodel and from
its extension. For instance, we can see the *markings* reference that have been
added within the *PetriNet* class or the *Token* class added within the *PetriNet*
package. We are identifying common concepts in KM3 metamodels by using a

```
1 package PetriNet {
2   class PetriNet {
3     reference elements[1-*] container : Element oppositeOf net;
4     -- @begin extensionOf
5     reference markings[*] : Marking;
6     -- @end extensionOf
7   }
8   abstract class Element {
9     attribute name : String;
10    reference net : PetriNet oppositeOf elements;
11  }
12  abstract class Node extends Element {
13    reference in[*] : Arc oppositeOf to;
14    reference out[*] : Arc oppositeOf from;
15  }
16  class Arc extends Element {
17    reference from : Node oppositeOf out;
18    reference to : Node oppositeOf in;
19  }
20  class Place extends Node {}
21  class Transition extends Node {}
22  -- @begin extensionOf
23  class Marking {
24    reference petrinet : PetriNet;
25    reference tokens[*] container : Token;
26  }
27  class Token {
28    reference position : Place;
29  }
30  -- @end extensionOf
31 }
```

Fig. 12. Result of PetriNet metamodel and marking extension combination in KM3

fully qualified name. The elements that are added by the extension metamodel
are surrounded by the – *@begin extensionOf* and – *@end extensionOf* comments.

4   Conceptual Framework

The initial limitation of our implementation handled KM3 metamodel extension
by concatenating the textual representation of the the participating metamod-
els. The new implementation of metamodel extension is focused on the abstract
syntax and has been fully automatized using model transformation. A first trans-
formation matches model elements by name, and a second one uses this mapping
as input to merge the initial and extension metamodels.

The motivating example introduced in the previous section described the con-
cept of model extension informally. Some examples of implementation have then
been given with the KM3 notation. This notation has been precisely defined within
a conceptual framework in a previous work [5]. The first two following definitions
come from this conceptual framework. They are repeated here for convenience. The
last definition extends those two to describe model extension formally.

Definition 1. *A directed multigraph $G = (N_G, E_G, \Gamma_G)$ consists of a finite set
of nodes N_G, a finite set of edges E_G, and a mapping function $\Gamma_G : E_G \rightarrow
N_G \times N_G$ mapping edges to their source and target nodes.*

Definition 2. *A model $M = (G, \omega, \mu)$ is a triple where:*

- $G = (N_G, E_G, \Gamma_G)$ *is a directed multigraph,*
- ω *is itself a model (called the reference model of M) associated to a graph $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$,*
- $\mu : N_G \cup E_G \to N_\omega$ *is a function associating elements (nodes and edges) of G to nodes of G_ω.*

Definition 3. *Let M_i and M_f be two models conforming to the same reference model ω*

- $M_i = (G_i, \omega, \mu_i)$ *and* $G_i = (N_i, E_i, \Gamma_i)$
- $M_f = (G_f, \omega, \mu_f)$ *and* $G_f = (N_f, E_f, \Gamma_f)$
- $\omega = (G_\omega, \omega_\omega, \mu_\omega)$ *and* $G_\omega = (N_\omega, E_\omega, \Gamma_\omega)$

Let $\epsilon : N_f \to N_f \cup N_i$ be a mapping from nodes of M_f to nodes of M_f and nodes of M_i. ϵ maps each node from N_f to that same node or to a node from N_i:

$$\epsilon(x) = \begin{cases} x & \text{if } \forall v \in N_i, v \neq x \text{ or,} \\ y & \text{if } \exists y \in N_i, y = x \end{cases}$$

Note: Node comparison operators $=$ and $\neq$ must be defined in a metamodel specific way. For instance, in Sect. 3, classes (KM3 specific concept) were compared according to their names. The definition of such operators is out of the scope of this paper. We are considering this comparison as a decidable and deterministic issue in the following definitions.

The extension $M_r = M_i \oplus_\epsilon M_f$ of model M_i by model M_f according to ϵ is the model M_r with

- $M_r = (G_r, \omega_r, \mu_r)$ *and* $G_r = (N_r, E_r, \Gamma_r)$
  - $N_r = N_i \cup \epsilon(N_f)$,
  - $E_r = E_i \cup E_f$,
  - $\Gamma_r(x) = \begin{cases} \Gamma_i(x) & \text{if } x \in E_i \\ \epsilon^{\{2\}}(\Gamma_f(x)) & \text{if } x \in E_f \end{cases}$,
  - $\mu_r(x) = \begin{cases} \mu_i(x) & \text{if } x \in N_i \cup E_i \\ \mu_f(x) & \text{if } x \in N_f \cup E_f \end{cases}$.

The function $\epsilon^{\{2\}} : N_f^2 \to (N_f \cup N_i)^2$ is the bidimensional version of $\epsilon : x \mapsto y$ defined as $\epsilon^{\{2\}} : (x, y) \mapsto (\epsilon(x), \epsilon(y))$

An illustration of model extension is given in Fig. 13. The initial model M_i containing four elements is extended by M_f containing three elements according to the ϵ mapping. ϵ maps the two lower nodes of M_f to themselves and the third one to a node of M_i. The result of the extension is M_r, which contains six nodes: all the four from M_i and only two from M_f.
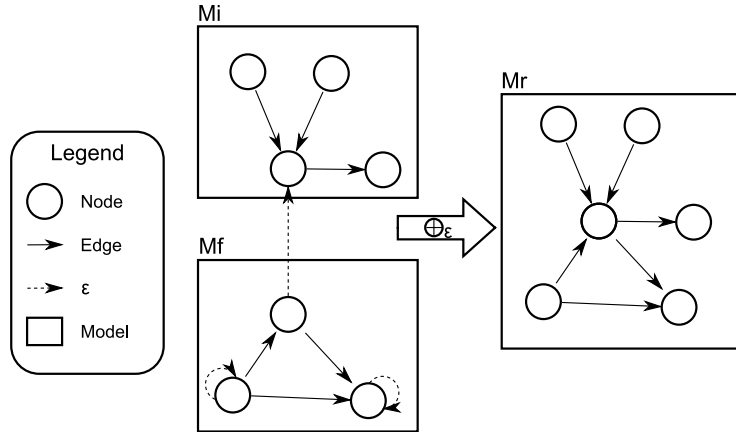
Fig. 13. Example of model extension

5   Characterization of Model Extension

From previous definitions, some characteristics of model extension may be leveraged. The following (non-exhaustive) list presents some of the characteristics of use for model extension.

- Fragment model has no dangling edges. A dangling edge is an edge linked to only one node. With this kind of structure, a fragment model would have been a special entity. Because there is no dangling edge, the fragment model is a "true" model conforming to a reference model.
- Models M_i, M_r and M_f conform to the same reference model. Even the fragment model M_f is a model conforming to the reference model. This result comes from the absence of dangling edge.
- With our conceptual definitions, it is possible to define libraries of model extensions. These libraries should have a lattice structure. For instance, we could have a library of Petri net metamodels, with each metamodel capturing the concepts of each type of Petri net by extending another kind of Petri net metamodel.

6   Related Work

The problem of model extension is central to most practical model editing tasks. Many solutions have been found in specific contexts. For intance, in case one is dealing only with UML models (i.e., models conforming to the UML metamodel), then specific ad-hoc solutions based on profiles have been proposed [3].

There have been a lot of discussions comparing these ad-hoc UML profiling techniques and heavyweight metamodel based solutions considering MOF-conforming metamodels. According to the specific practical context, it has been

found that one solution is better than the other for reasons of tool availability [13,14,11,4,6].

Considering model extension, there is a similar principle in the UML 2 Infrastructure specification [10] called package merge. This operation was defined to assist in modularizing the UML 2 metamodel. It also defines compliance levels regarding the packages that are merged. We chose not to follow this specification for several reasons. First, it is an UML-specific operation that is difficult to express for other kinds of metamodels. Second, the package merge operation has not been provided in a clearly defined conceptual framework. Finally, it has been shown that UML 2 package merge has many problems and can not be used in its present definition [15,16].

In this paper, we propose a more general approach. To this purpose, we have chosen the KM3 minimal notation to build a conceptual and practical solution. This has several advantages. First, since there are significant libraries of open source metamodels in KM3, this will allow direct experimentation on the basis of the available metamodels. Second, there are available bridges between KM3 and most popular metametamodels like MOF 1.4, MOF 2.0, Ecore, MetaGME, and Microsoft DSL Tools. As a consequence, the extensibility solution proposed here could be mapped to these other representation systems. Finally, since KM3 is a minimal metametamodel, this permits a basic conceptual solution that has more chances to be independent of implementation idiosyncrasies.

7   Conclusions and Future Work

This paper introduced the *extensionOf* relation between models. This relation has been formally defined within a conceptual framework. Definitions given in Sect. 4 are generic and do not rely on a specific metamodel. The solution introduced in this paper is being implemented as part of the AMMA model engineering platform under Eclipse. This approach will lead to capabilities that assist in the composition of models.

Metamodel extensibility is very important to define auto-adaptive tools by coupling a core metamodel extension and a set of base tool extensions. The Atlas Model Weaver [2] is already a proof of concept of this approach. It defines a core weaving metamodel and a generic editor. The editor auto-adapts itself depending on the metamodel extensions specified by the user. Moreover, it is also possible to define extensions to each editor part for a specific metamodel extension.

Finally, following the principles presented in [7], Domain-Specific Languages (DSLs) [9] extension could be based on model extension. We are considering a DSL as a set of coordinated models [7]: an abstract syntax, a concrete syntax, and a specification of the semantics. From a model-based point of view, the abstract syntax of a DSL is a metamodel. Thus, DSL concepts can be extended with the mechanism described in this paper. The study of how to extend the concrete syntax model and how it is related to grammar extension represents areas of future work.

Acknowledgements

References

1. Bézivin, J.: On the Unification Power of Models. Software and Systems Modeling 4(2), 171–188 (2005)
2. Didonet Del Fabro, M., Bézivin, J., Valduriez, P.: Weaving Models with the Eclipse AMW plugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006, Esslingen, Germany (2006)
3. D'Souza, D., Sane, A., Birchenough, A.: First-Class Extensibility for UML Packaging of Profiles, Stereotypes, Patterns. In: France, R.B., Rumpe, B. (eds.) UML '99 - The Unified Modeling Language. Beyond the Standard. LNCS, vol. 1723, Springer, Heidelberg (1999)
4. Gitzel, R., Hildenbrand, T.: A taxonomy of Metamodel Hieriarchies, University of Mannheim (January 2005)
5. Jouault, F., Bézivin, J.: KM3: a DSL for Metamodel Specification. In: Gorrieri, R., Wehrheim, H. (eds.) FMOODS 2006. LNCS, vol. 4037, Springer, Heidelberg (2006)
6. Karsai, G., Maroti, M., Ledeczi, A., Gray, J., Sztipanovits, J.: Composition and Cloning in Modeling and Meta-Modeling Languages, IEEE Transactions on Control System Technology, special issue on Computer Automated Multi-Paradigm Modeling, 263–278 (March 2004)
7. Kurtev, I., Bézivin, J., Jouault, F., Valduriez, P.: Model-based DSL Frameworks. In: Companion to the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, Portland, OR, USA (October 22-26, 2006) 602–616 (2006)
8. Liskov, B., Wing, J.: A Behavioral Notion of Subtyping. ACM Transactions on Programming Languages and Systems 16(6), 1811–1841 (1994)
9. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. 37(4), 316–344 (2005)
10. Object Management Group: Unified Modeling Language: Infrastructure, version 2.1.1, formal/07-02-06, `http://www.omg.org/cgi-bin/doc?formal/07-02-06`
11. Pérez-Martínez, J.E.: Heavyweight extensions to the UML metamodel to describe the C3 architectural style. ACM SIGSOFT Software Engineering Notes 28(3), 5 (2003)
12. Peterson, J.: Petri Nets, ACM Computing Surveys, 223–252 (September 1977)
13. Rötschke, T.: Adding Pluggable Meta Models to FUJABA, In: 2nd International Fujaba Days, 2004, 04-253, Universität Paderborn, 57–61 (2004)
14. Turki, S., Soriano, T.: A SysML Extension for Bond Graphs Support ICTA'05, 5th International Conference on Technology and Automation, Thessaloniki, Greece, 276–281 (October 2005)
15. Zito, A., Diskin, Z., Dingel, J.: Package Merge in UML 2: Practice vs. Theory?, Model Driven Engineering Languages and Systems, 185–199 (2006)
16. Zito, A., Dingel, J.: Modeling UML 2 Package Merge With Alloy, In: Proc. of the 1st Alloy Workshop (Alloy '06). Portland, Oregon, USA (Nov (2006)