A Two-tiered Methodology to Extend the UML Metamodel

Josep M. Ribó[1], Xavier Franch[2]

[1] Universitat de Lleida
C. Jaume II, 69 E-25001 Lleida (Catalunya, Spain)
josepma@eup.udl.es
[2] Universitat Politècnica de Catalunya (UPC),
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
franch@lsi.upc.es

Abstract. The usage of UML in specific contexts (like real-time systems or process modelling) is specially appealing since it provides a standard modelling notation widely used by the software engineering community. However, such usage usually requires to tailor (extend) the UML metamodel. The standard extension mechanisms, although enhanced in UML v.1.4, still suffer from several expressiveness limitations. In this article we identify these limitations and we define a two-tiered methodology to construct standard metamodels as extensions of the UML metamodel. Specifically, we present a methodology to extend explicitly the UML metamodel (as a particular case of MOF-model instance) and another to transform an extended UML metamodel into a UML profile. By means of these methodologies, we are able to combine the expressiveness provided by the explicit extension with the standardization coming from the use of profiles, which allows also the usage of existing tools.

1. Introduction

In the last few years, UML has emerged as a standard multi-purpose modelling language, widely used within the software engineering community. But it is obvious that, regardless of its quality, it is difficult for a single modelling language to meet the requirements of every particular modelling domain; for instance, many differences appear when considering modelling of real-time systems and software processes. It seems clear that each modelling domain will have specific needs and that some extension of the UML metamodel will be necessary to address those needs appropriately.

UML metamodelling strategy is based on the 4-layer metamodelling architecture [MOF00], which has been adopted by the OMG and which establishes four metamodel levels: M3 (meta-metamodel), M2 (metamodel), M1 (specific model) and M0 (user objects). According to this architecture, the responsibility of the layer M_j is to define the language that will be used to describe the objects of the layer M_{j-1}. Put it another way: a model in the layer M_{j-1} is an instance of a model in the layer M_j. The UML metamodel is defined at level M2. The MOF model is the UML meta-metamodel and it is defined at level M3.

There exist several approaches to extend the M2 UML metamodel, which have different features. In this article, we outline those approaches, we discuss some of their properties and we propose an extension methodology which maximizes the binomial *expressiveness-standardization* and which keeps a good degree of readability.

In particular, our approach consists in (a) presenting an algorithm to build an additive extension of the UML metamodel, while keeping some properties that guarantee the semantic consistency of the extension¸and (b) presenting a procedure that transforms that metamodel extension into a UML profile.

This two-tiered approach keeps (a) the expressiveness and readability that may be reached by means of an explicit metamodel extension and (b) the standardization provided by the UML profiles.

The results presented in this paper have been obtained taking advantage of the experience gained during the development of the (UML-based) PROMENADE process modelling language in the field of software process modelling [FR99, RF01]. We realized that our methodology is suitable for UML extensions in different domains and, in fact, throughout the article, we provide examples coming from the e-commerce context.

2. UML metamodel extensions

In the literature, three different alternatives have been developed to define UML metamodel extensions. In this section, we stress their advantages and drawbacks with respect to several features, namely: expressiveness; standardization; readability; robustness with respect to new UML releases; and conformance to the 4-layer architecture. Next, we present our own approach trying to optimize standardization, expressiveness and readability altogether, while keeping robustness and conformance.

1. To provide a direct instantiation of the MOF model

This approach consists in constructing a metamodel conforming to the MOF model. The resulting metamodel will be a *strict* or *loose* instance of the MOF model (an instance is strict if each of its elements is an instance of a MOF model one). This was the initial approach taken by UPM [UPM00] in the context of software process modelling.

- *Expressiveness*. This is clearly the most powerful approach to construct a metamodel, since conformity with the UML metamodel is not required.
- *Standardization*. The resulting metamodel is not actually an extension of UML; therefore, we cannot rely on the UML semantics nor can we use directly UML-tools and even the own UML notation. This drawback holds even if as many UML metaelements as possible are kept in order to maintain a similar notation with similar semantics.
- *Readability*. The extended metamodel can be expressed in a single and uniform representation, i.e., a representation bound to the M2 level of the 4-layer architecture.
- *Robustness*. It is robust in front of changes in the UML metamodel, since it is not supposed to be consistent with it.

- *Conformance*. Strict metamodeling guarantees a full conformance to the 4-layer metamodelling architecture; loose metamodelling provides model-granularity conformance but not element granularity conformance (see above).

2. To construct a metamodel by derivation of the UML metamodel

This approach, usually referred to as *heavyweight extension*, consists in creating an explicit extension of the UML metamodel in an additive way (i.e., adding new metaelements without altering the semantics of the existing ones). It is the approach taken, among others, by [SPE01] in the context of software process modelling (substituting the UPM cited above) and CWM [CWM00] in data warehouse.

- *Expressiveness*. Heavyweight extension is a bit more restrictive than direct MOF extension, because conformance to the semantics of the existing classes in the UML metamodel is required. However, it is still quite powerful because it is allowed to add new metaelements to that metamodel (by subclasifying the existing ones).
- *Standardization*. Heavyweight extensions allow to rely on the semantics of UML and to use straightforwardly its existing notation and also UML-based tools for the pre-existent UML elements. However, some new semantics and notation to represent graphically the added elements are needed, which compromises the standardization of the approach.
- *Readability*. The extended metamodel can be expressed in a single and uniform representation.
- *Robustness*. Changes on those parts of the UML metamodel involving the extension being defined affect heavyweight extensions.
- *Conformance*. Since the UML metamodel is a loose instance of the MOF model this approach is restricted to element granularity conformance.

3. To define a UML profile

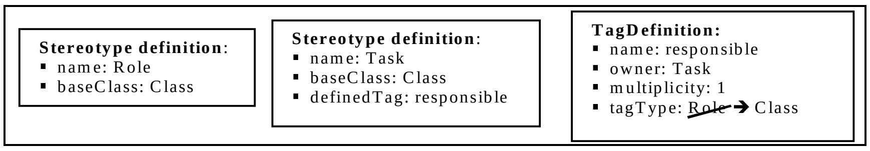
UML profiles, also known as *lightweight extensions*, are based on the use of the UML built-in extension mechanisms in order to specialize UML for a concrete domain. In particular, a UML profile is created by defining *stereotypes*, *tagged-values* (based on *tag definitions*) and *constraints* on the UML metaclasses. UML profiles are meant to be purely additive extensions of the UML metamodel. Many UML profiles have been defined. See, for instance [SPE01, UML01].

Although UML v.1.4 [UML01] has improved the extension mechanisms by means of a more rigorous definition of tagged-values, this approach suffers from several limitations that compromise the expressiveness and correctness of the extended metamodel. We summarize in the following some of these limitations:

(1) New metaelements defined by using the UML extension mechanisms do not have the same semantics as UML actual metaclasses. They are defined as instances of the *Stereotype* UML metaclass. Therefore, they are actually M1 elements without either an identity or a representation as M2 elements. As a result, UML profiles only simulate metaelements and their features, which are in M2, and they cannot be integrated into the UML metamodel representation, compromising readability.

(2) Extension mechanisms allow the definition of *pseudometaclasses or pseudoattributes* but they do not offer straightforward ways to define new associations, dependencies and class operations.

Another limitation comes with tag definitions: a tag definition may not have a specific stereotyped metaclass as *tagType*. Constraint #1 of *TagDefinition* [UML01, p. 2-82] states that *tagType* may be the name of a UML metaclass. According to this constraint, *Stereotype* is a valid *tagType*. However, a specific instance of *Stereotype* is not. As a consequence, definition of references to other stereotypes is not straightforward. In fig. 1, we show this problem: we have defined two stereotypes: <<*Role*>> and <<*Task*>> with base class *Class;* now we want to associate a *TagDefinition* to the stereotype <<*Task*>> to establish that a given task class should have a role as responsible. The *tagType* of the *TagDefinition* should be "Role" but this is not possible since "Role" is not the name of any UML metaclass. Hence, the tagType becomes *Class.* Some additional constraints will have to be stated.



Stereotype definition:
- name: Role
- baseClass: Class

Stereotype definition:
- name: Task
- baseClass: Class
- definedTag: responsible

TagDefinition:
- name: responsible
- owner: Task
- multiplicity: 1
- tagType: ~~Role~~ $\rightarrow$ Class

Fig. 1: Association of a TagDefinition to a stereotype

(3) We have found some ambiguities in the semantics of the extension mechanisms. For instance, there is no constraint restricting the number of reference values associated to a specific tagged value to be exactly the multiplicity established by its corresponding tag definition.

As a second example, tagged values which have a *DataType* as *tagType* are not real values from that *tagType* but just strings. The conformance between the actual value and the type is ambiguous and difficult to check. In fact, this may lead to inconsistencies between the attribute type and the value that it actually stores.

Some of these drawbacks (namely (2) and (3)) could be resolved in future UML specifications. However, the semantic mismatch (1) brought up by profiles are inherent to their own definition and compromise the 4-layer metamodelling architecture.

We sum up the behaviour of this approach with respect to the considered features:

- *Expressiveness*. Several important elements (such as dependencies or associations) cannot be directly expressed in a UML profile.
- *Standardization*. Profiles are based on the extension UML mechanisms. Therefore, they provide a full standardization with respect to UML .
- *Readability*. There is no representation for the extended metamodel elements at level M2. Furthermore, the construction mechanisms for the extension are different from the ones used for defining the UML metamodel (see drawback (1) above).
- *Robustness*. In addition to changes that affect heavyweight extensions, those involving UML extension mechanisms also affect profiles.

- *Conformance*. Profiles challenge the 4-layer metamodelling architecture. It could be a matter of discussion whether the UML metamodel extended with a profile is an instance of the MOF model. Our particular position is that it is not.

4. Our proposal

In this paper we propose a methodology based on the combination of heavyweight and lightweight extensions. Building, first of all, a heavyweight extension, we obtain an expressive and well-defined metamodel. Transforming this metamodel into a UML profile, we obtain a metamodel for a specific domain, which is fully standard (i.e., available in standard UML). Due to the existence of these two alternative metamodels, we call this methodology *two-tiered*. The methodology is defined as the composition of the following steps (see fig. 2):

1. *Restriction*. Those metaelements which are not of interest in the actual modelling context are discarded from the UML metamodel. We call *restricted metamodel* the result. We remark that this step is not offered by heavyweight extension; as a consequence, although the metamodel is intented to be tailored to a specific domain, in fact it includes elements which are not necessary and that difficult the readability of the metamodel.
2. *Extension*. The restricted metamodel is extended to meet the specific requirements of the modelling context. We call the resulting model *extended metamodel.* The extended metamodel will be an instance of the MOF model with an additive definition with respect to the UML metamodel. Therefore, it is assured that the semantics of the remaining UML metaelements will not be affected in any way.
3. *Transformation*. The extended metamodel is transformed into a UML profile defined according to UML v.1.4. The resulting *transformed metamodel* offers a complete standardization and can be manipulated with general UML modelling tools that support profiles, such as the *Objecteering / UML profile builder* [Obj02].

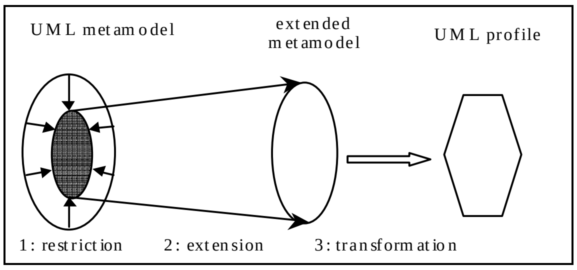


Fig. 2: A two-tiered extension of the UML metamodel

The main advantage of our two-tiered approach is that we use the appropriate metamodel in the adequate context, that is: the extended metamodel for first defining the extension, maintaining it and reasoning about it, while the transformed metamodel (i.e., the UML profile) is used for model definition (instantiation) and portability (which allows using existing UML tools). If we analyze the five criteria used so far:

- *Expressiveness*. The extended metamodel may be defined with all the expressiveness supplied by the MOF model (in particular, in contrast with the profile approach, associations, dependencies, real features and other elements may have been incorporated to it in a natural way).

- *Standardization*. The transformed metamodel, which is obtained after a well-defined procedure (see section 4) applied to the extended metamodel, guarantees full compatibility with standard UML.
- *Readability*. The extended metamodel is completely represented at level M2. This provides a good degree of readability, which facilitates the metamodel development and its maintenance if changes are to be incorporated in the future.
- *Robustness*. Changes on those parts of the UML metamodel involving the extension being defined affect the extended metamodel. Changes on the UML extension mechanisms affect the transformed metamodel.
- *Conformance*. The extended metamodel conforms completely with the 4-layer metamodeling architecture.

In particular, the expressiveness, readability and conformance drawbacks we have detected in profiles (see section 2.3) are not as rellevant as working with profiles directly because the rellevant metamodel with respect to these criteria is the extended metamodel.

The price to pay in our approach is the extra effort needed for transforming the extended metamodel into a UML profile. However, it could be argued that even the direct construction of a UML profile as done in 2.3 is preceded by an implicit definition of a conceptual metamodel; our proposal just makes it explicit. Furthermore, the effort may be substantially lowered by using a tool for supporting such a transformation, taking advantage of the existence of a well-defined methodology, presented in section 4.

It is clear that our two-tiered approach brings up some important challenges and questions. What is a UML metamodel restriction? How can it be created? How a UML metamodel can be extended in an additive way without compromising the UML underlying semantics? How can it be transformed into a UML profile without losing its expressiveness? We try to answer this questions in the next few sections.

3. Explicit restriction-extension of the UML metamodel

In this section we present a methodology to perform a restriction and then an extension of the UML v.1.4 metamodel in such a way that it is still an instance of the MOF model. This methodology is not restricted to the UML metamodel. In fact, it may be applied to perform a restriction-extension of any MOF model instance.

3.1. Restriction of an instance of the MOF model

Definition. Let *m* and *mr* be two instances of the MOF model. We say that *mr* is a restriction of *m* iff:

(1) All the elements in *mr* are also in *m*.
Notice that the elements of an instance of the MOF model may be either instances of MOF classes or instances of MOF associations (also called *links*).

(2) *mr* must be self-contained.

That is: let *a* be any *m*'s MOF association instance such that *a* links an instance of a *mr*'s MOF class with some other association-end *x*. In this situation both *x* and *a* should come up at *mr*.

The condition (2) has several implications. For instance, any *mr*'s MOF class instance which is not the root, must have its supertype in *mr*. Other similar conditions for the different types of MOF model associations exist (see below).

Next we present a procedure to generate a restriction of the UML metamodel (hereafter, *UML-M*). The restriction should contain the set of elements *E* of UML-M that we want to reuse in the metamodel under construction. According to the definition of restriction, it should also incorporate transitively the elements linked to those in *E*. Specifically, the restriction *UML-R* of UML-M is defined in the following way:

(1) Let $E0 = E$.

(2) Given a set E_k, $k \geq 0$, we define E_{k+1} the minimum set such that:

— $E_k \subseteq E_{k+1}$

— E_{k+1} includes all the elements linked to *e* by means of MOF model associations:

$$\forall e \in E_k: \exists x: \Re\ (x, e) \Rightarrow x \in E_{k+1},$$

being $\Re$ any of: *Generalizes*, *Aliases*, *Contains*, *AttachesTo*, *RefersTo*, *CanRaise*, *IsOfType*, *Constraints*, *DependsOn*.

(3) $UML\text{-}R = \min k: k \geq 0: E_k = E_{k+1}$

3.2. Extension of an instance of the MOF model

Extending an instance of the MOF model (such as the UML metamodel) involves adding new metaelements to that instance. Each one of those metaelements should be an instance of a meta-metaelement (either a class or an association) defined in the MOF model [MOF00], including not only classes and attributes but also dependencies, associations, etc., which are not defined in UML profiles. This addition must take into account that the MOF model states some restrictions to be kept by its instantiations. For example: only binary associations are allowed between classifiers; MOF model containment hierarchy should be respected; etc.

We will extend an instance of the MOF model by adding instances of the abovementioned metaelements in such a way that (a) they keep the restrictions stated by the MOF model and (b) they do not alter the semantics of the departing metamodel.

Definition. Let *m* and *mext* be two instances of the MOF model. We say that *mext* is an additive extension of *m* iff:

(1) All the metaelements that *mext* adds to *m* are instances of meta-metaelements from the MOF model and the restrictions defined by the MOF model are kept.

(2) *mext* contains all the elements that belong to *m*.

(3) No *m*'s element has been modified in any way within *mext* (therefore, *m*'s semantics is preserved).

Condition (3) implies the following restrictions (which are deduced from the definition of the MOF model): (a) A *mext*'s element that is originally defined in *m* (i.e. comes from *m*) must have as its supertype an element coming from *m*. (b) An *mext*'s element that comes from *m* must not depend on an element that does not come from *m*. (c) The elements contained by any *m*'s element must not change in *mext*. (d) No new constraints can be defined on an *m*'s element. Moreover, a constraint that comes from *m* cannot be applied to some new elements in *mext*. (e) No new imports can be defined for an element that comes from m. (f) No composite aggregation such that either the composite or the component classes comes from *m* can be defined. (g) No new associations between classes coming from *m* may be defined. (h) Any association *a* defined in *mext* such that *a* links a class *c* from *m* with another class *d* not in *m*, will be defined in such a way that the association-end opposite to *c* must have its *isNavigable* attribute set to false (i.e., *a* must be oriented to the class that comes from *m*).

Although conditions (g) and (h) are not strictly necessary to be kept, they are certainly convenient. According to the MOF model, the definition of a new association *a* between an *m*'s class *c* and another class *d* not in *m* does not modify *c*'s definition. It simply adds a new instance of the MOF class *Association* and two instances of the MOF class *AssociationEnd* contained in the former. Therefore, *m* is not modified by *a*'s definition. However, if *c*'s opposite end is made navigable, *c*'s semantics change implicitly with respect to *m*.

Figure 3 shows an example in the context of building a metamodel for e-commerce process modelling. We extend the UML metamodel with the metaclass *eCPM* (which stands for a model of an e-commerce process) and we want to state that the static part of an e-commerce process model may consist, among other things, of several (UML) generalizations between some of its constituents (i.e., to create taxonomies of products or hierarchies of activities). This may be modelled as in figure 3. If navigation from *Generalization* to *eCPM* had been allowed, this would have altered implicitly the UML semantics since it does not make any sense, in the context of UML, to refer to the instances of *eCPM* associated to a generalization. Notice that, although *c*'s definition does not change at level M2, the representation of the association *a* will be made, in all probability, by means of a *reference* to *d* stored by *c*.

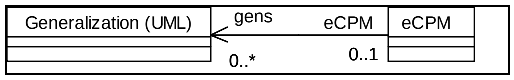


Fig. 3: Extending the UML metamodel with an association

Last, we remark that some of the above-mentioned restrictions may be overcome by including a subclassification of *m*'s elements into *mext*. For instance, although an extension of a model *m* may not define new constraints on a *m*'s element, this element may be subclassified and, hence, some constraints may be defined for this subclass.

3.3. Extension algorithm

In order to generate an extension *UML-E* from the restricted model *UML-R* we carry out the following activities:

1. *Identify the set of all the metaclasses that should be contained in UML-E which either have no correspondence in UML-M or specialize significatively some element in UML-M. Call newels to that set.*

We want to create a *minimal* extension. Therefore, for each element *e* in *newels*, it is important to justify that there is not any element in the UML metamodel into which *e* could be assimilated and that the specialization offered by this element is significative enough.

However, this *minimality criterium* should be applied carefully. Sometimes, the reuse of an existing UML metalement may lead to inconsistent overlaps. Consider the following situation in the context of e-commerce process modelling. Instances of the new metaclass *Task*, which represent activities carried out during e-commerce transactions, may have parameters, which represent documents (e.g., product to be bought) or data (e.g., client ID or money amount). Hence, we decide to reuse the *Parameter* UML metaclass and we define an association between *Task* and *Parameter* as in fig. 4. However, this association overlaps with the aggregation in UML-M that establishes that a *Parameter* should belong to a *BehaviouralFeature*. Therefore, a task (which is not a behavioural feature) cannot have UML parameters. In this case, the reuse of *Parameter* leads to an inconsistency and should be avoided.

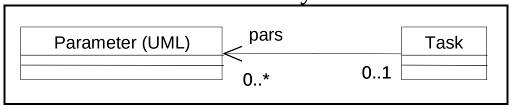


Fig. 4: An inconsistent reuse of the *Parameter* metaclass

2. Integrate each element e in newels into a generalization hierarchy.
In order to do that, for each element *e* in newels, select either the closest element *e'* from UML-M such that *e* specializes *e'* or some other element *e''* in *newels* such that *e* specializes *e''*. Notice that, in this way, we enforce the restriction (a) that no UML-M element will have a *newel* as superclass.
3. *Elaborate the containment hierarchy within newels.*
Recall that the UML-M containment hierarchy cannot be altered. Therefore, new containments may only be defined between elements belonging to *newels.* In addition, some new packages may be defined in order to group some related elements from *newels.* Some constraints and imports may be associated to these new defined packages, which may include UML-M elements (i.e., these elements are not modified).

Notice that the MOF model does not define any restriction that forces generalizations, associations or dependencies to be established between elements belonging to the same namespace.
4. Identify those associations that involve some element e in newels
These associations will be defined between exactly two elements in *newels* or between one element in *newels* and another one in UML-M. Some constraints may be defined on these associations.

In order to keep the restrictions (f), (g) and (h) presented in section 3.2, no composite aggregation involving UML-M elements will be defined. On the other hand, all the associations involving a UML-M element will be oriented to that element. All the defined associations will be binary.

In application of the criterium of minimal extension, a new association should be added only if it is strictly necessary. It is preferable to avoid redundancies by reusing/adapting or even replacing (see [SW01]) existing UML-M association to the new necessities. The adaptation may be done by means of constraints that restricts the new use of the association (see exemple in section 3.4). We will see in section 4.2 that the reuse of UML-M associations facilitates the transformation of UML-E into a UML profile (there is no direct way to transform associations into a UML profile). However, as we have shown in (1), reuse of UML-M elements must be done carefully.

5. *Identify those dependencies existing between two elements in newels or from one element in newels which depends on another one in UML-M.*

Notice that, in order to keep restriction (b), for any dependency involving a UML-M element, the provider of the dependency should be this UML-M element.

6. *Provide a definition for each class c in newels.*

Such definition will consist of:

- a list of attributes for *c*.
- a list of operations for *c*. These operations may enumerate a list of exceptions raised by them and several parameters which must have as type some classifier that belongs to the set of UML-M classifiers ∪ *newels*.
- a list of references corresponding to some of the associations involving *c* (the associations involving *c* are those associations *a* such that *c* is an association-end of *a*). Although it is not necessary to define references for each association involving *c*, it may be useful for the algorithm that transforms an extended UML metamodel into a UML profile. We will turn back to this idea in section 4.2.
- a list of constraints associated to *c*.

7. *If necessary, add to the model some elements like tags, constants and data types.*

Notice that the restriction-extension algorithm may be applied in an iterative way: if during the extension step we become aware of the need of some UML-M element that was not included in UML-R, we may start it over.

3.4. Example

In this section we outline an extension example: the incorporation of *precedence relationships* (*precedences,* for short) to the UML metamodel.

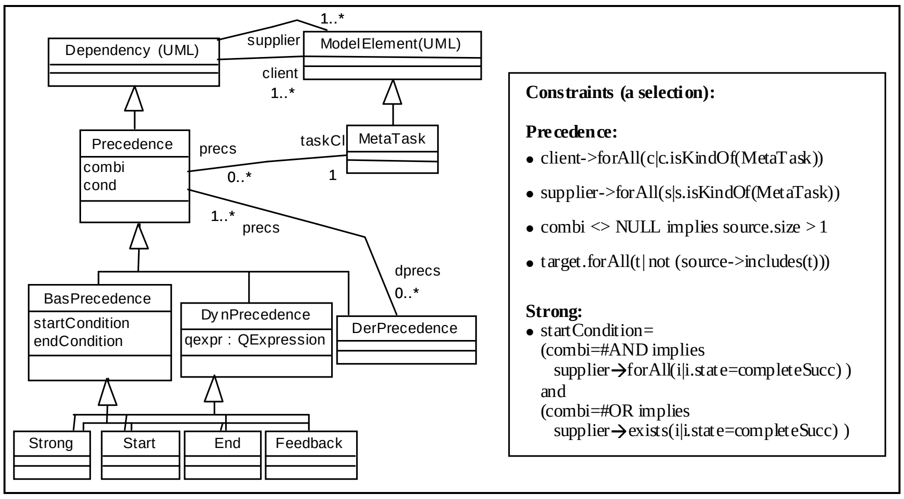
Precedences come up in various contexts; for instance, one of the key points in establishing models of e-commerce is stating the temporal precedences between the different activities that take part in these processes. Temporal precedences allow the arrangement of activities and time, supporting then the precise statement of models. We may find many different types of temporal precedences between activities. For example, a component delivering information to 10.000 subscriber agents should not be waiting until completion before performing other activities; on the other hand, during a peer-to-peer negotiation, activities must be strictly sequencialized.

Many approaches in similar domains, remarkably workflow technology and software process modelling, introduce the concept of precedence explicitly in their modeling formalism [JB96, JPL98, RF00]. A precedence is stated between a set of source task classes and another set of target ones and establishes in a *declarative* way which requirements (concerning the state of the source tasks) are needed in order to start/finish the enactment of the target ones. In addition, precedences make explicit the binding between the documents and other data that are involved in these tasks by means of links between task parameters. The proactive behaviour of a specific composite task is stated by means of a collection of precedences between its subtask classes.

We have previously shown in [RF00] that this concept of precedence is conceptually different from that of UML transitions. Therefore, we cannot use the UML transitions/activity diagrams in order to model precedences. Instead, we will extend the UML metamodel with the metaclass *Precedence*, which will be incorporated as a subclass of the closest metaelement within the UML metamodel: *Dependency.*

According to the UML metamodel, a dependency states that *the implementation or functioning of one or more elements requires the presence of one or more other elements* [UML01, p. 2-33]. Therefore, a dependency is modelled as a relationship between a (set of) client and a (set of) supplier meaning that the client is dependent on the supplier. It seems quite natural to represent a precedence as an behavioural dependency from a (set of) client activities to a (set of) supplier activities meaning that the the enactment of the client activity depends on that of the supplier activity.

As it is shown in figure 5, the e-commerce metamodel defines several families of precedences which leads to a hierarchy of new metaclasses. Basic precedences are the ones described in terms of task states, while derived precedences are defined in terms of other precedences. By default, precedences involve a fixed number of participants, but we add dynamic precedences for modeling a variable number of them.



Constraints (a selection):

Precedence:

- client->forAll(c|c.isKindOf(MetaTask))
- supplier->forAll(s|s.isKindOf(MetaTask))
- combi <> NULL implies source.size > 1
- target.forAll(t| not (source->includes(t)))

Strong:

- startCondition=
  (combi=#AND implies
    supplier→forAll(i|i.state=completeSucc))
  and
  (combi=#OR implies
    supplier→exists(i|i.state=completeSucc))

Fig. 5: UML metamodel extended to deal with precedences (fragment)

Notice that we reuse the UML association linking *Dependency* with *ModelElement* to state clients and suppliers of the precedence. However, we adapt the semantics of that association to precedences by establishing (using constraints associated to *Precedence*) that *clients* and *suppliers* of precedences must be task classes. We also add some constraints to restrict some other aspects of precedences.

4. Transformation to a UML-profile

In this section we present a methodology to transform a metamodel extension constructed following the procedure presented in section 3 into a UML profile. We focus on the most used metaelements to be transformed (namely, classes, data types, generalizations, attribute and association).

A metamodel *m* obtained using the restriction-extension algorithm presented above may be transformed into a UML profile by means of the following procedure:

1. Incorporate to the profile all the elements in *m* that come from UML-M.
2. Use the UML extension mechanisms in order to transform the remaining elements (i.e., those belonging to *m* but not to UML-M) into valid profile elements.

Obviously, step 2 is the most interesting one. In the following, we enumerate how the different elements that may constitute the extended metamodel can be transformed into valid elements of a UML-profile.

4.1. Class, generalization, constraint and attribute

For the sake of brevity we will not go into detail in the transformation of these metaelements, since they have a clear correspondence in profiles. In summary, classes will be transformed into stereotypes, using as base class its closest ancestor in UML-M. Attributes of a class *c* will be transformed into tag definitions, which will be associated to the stereotype into which *c* has been transformed to. Constraints may be added to the different extended elements to delimit their semantics.

4.2. Association

Contrary to the cases of metaclasses, generalizations and attributes, UML does not define any extension mechanism specifically intended to represent *pseudo*-associations. In this section, we present three alternative ways to transform M2 associations into valid elements in a UML profile. We will discard the first one, while admitting the other two.

1. Reuse of a UML association.

The idea is to transform an M2 association of the extended UML metamodel into one of the associations already existing in the UML metamodel. Therefore, it is not necessary to add any new element into the UML profile. This is the approach taken by [SPE01] in order to incorporate into a UML profile many of the new associations defined for the SPEM metamodel. We find two problems to this approach, the first one methodological and the second one semantical:

— As we have argued in section 3.3, new elements are to be added to UML-M only if they introduce new semantical concepts. If possible, associations already existing in the UML metamodel will be reused, adapted or replaced (see [SW01]). We prefer not to clutter the metamodel with redundant metaelements.

— For those associations fulfilling the previous condition, it will not be usually the case that they can be completely assimilated to another one of UML-M.

Some examples of redundant associations can be found in the SPEM metamodel [SPE01]. For instance, the SPEM association *WorkDefinition::owner* is redundant because there exist a UML association, *Feature::owner*, that does the same.

2. Use of references

Navigable associations may be represented by means of references. References are associated to the classifiers that act as the association-ends for a particular association and they refer to the classifier at the other end.

The MOF model allows the definition of references associated to the classifiers that participate in associations. These references may be transformed in a natural way into tag definitions and incorporated into a UML profile.

Therefore, we propose to accompany the associations defined into a UML metamodel extension with references in the association-end whose counterpart (opposite end) should be navigable (in both ends if both should be navigable).

Although this is an appropriate approach, it is not always applicable since it requires navigable associations. Therfore, we are committed to find another solution for this case.

3. Define stereotypes on the UML metaclass Association

For each association defined on a UML metamodel extension, we may create a stereotype (with base class *Association*). Some constraints may be defined on this stereotype in order to establish the classes that may act as association-ends for the stereotyped association. In the same way, some tagged-values can be associated to the stereotype to state multiplicity, navigability, etc.

At level M1 (model level), we can define instances of the stereotyped association between the classes that act as association ends for that particular association (according to the constraints defined). These links may be depicted in the usual UML style as lines between the class instances linked by the association accompanied by the stereotype.

Consider the following example. In a UML metamodel extension, we establish the association *is-responsible-for* between the metaclasses *Task* and *Role* (see figure 6). We consider that both association-ends are not navigable.

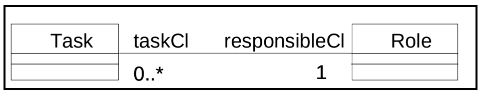


Fig. 6: The “responsibility” association in the extended metamodel

This association may be modelled in the context of a UML-profile as a specific stereotype (<<*ResponsibilityAss*>>) with base class *Association* . <<*Responsibil-ityAss*>> represents a special kind of association defined between task classes and role classes . Figure 7 contains the definition of the stereotype while figure 8 depicts an M1 model with the association.

- **Name:** <<ResponsibilityAss>> **Base class:** Association
- **Constraints** (applied to the stereotype <<ResponsibilityAss>>)**:**
  1. self.extendedElement->forall(a| a.connection->size=2)
  2. Let *mtst* be an instance of the <<Task>> stereotype.
     self.extendedElement->forAll(a|mtst.extendedElement->exists(t|a.connection->first.participant=t)
  3. Let *mrst* be an instance of the <<Role>> stereotype.
     self.extendedElement->forAll(a|mrst.extendedElement->exists(r| a.connection->last.participant=r)
  4. self.extendedElement->forall(a1,a2| a1<>a2 implies
     a1.connection->first.participant<>a2.connection->first.participant)

Fig. 7: Definition of the <<*ResponsibilityAss*>> stereotype

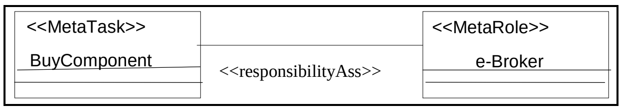


Fig. 8: An instance of a <<*ResponsibilityAss*>> association

The explicit extension of UML-M with a new association between a pair of metaclasses is not equivalent to the definition of a constrained stereotype on the base class *Association*, since the former association is defined at level M2 and the latter, at level M1.

However, constraints defined on the stereotype may help us to adapt the semantics of the stereotyped class to the intended one. In the case of <<*ResponsibilityAss*>> we establish that the association must be stated between exactly two classes (constraint 1); that the M1 association-ends must be <<*Task*>> and <<*Role*>>, respectively (constraints 2 and 3); and that only one M1 <<*ResponsibilityAss*>> association may be established having a particular task class as association-end (constraint 4). Due to this semantics deviation, we prefer the use of references (option 2) as a way to incorporate associations into a UML profile whenever possible.

4.3. Dependencies

The UML metamodel may be extended by the statement of new dependencies between metaelements. These dependencies may be mapped into a UML profile by defining a new stereotype <<*Dependant*>> on the base class *ModelElement* (from the UML metamodel). A tag definition is associated to this stereotype, which refers to the model element on which the stereotyped class depends. Figure 9 contains the definition of the stereotype <<*Dependant*>>.

- **Name:** <<Dependant>> **Base class:** ModelElement
- **Tag definitions:**
  name: dependant-on
  tagType: Stereotype
  multiplicity: *

Fig. 9: Definition of the <<Dependant>> stereotype

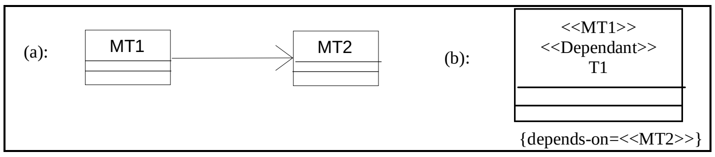


Fig. 10: Example of a M2 dependency and its transformation

Consider a dependency from the metaclass MT1 to MT2 appearing in the restricted-extended model (see figure 10(a)). From section 3.3 we can infer that MT1 will be a new metaclass that has been incorporated into the extended model and that M1 may be either a UML metaclass or a newly created one. In any case, a stereotype will have been created to MT1 in the process of construction of the UML profile. The dependency from MT1 to MT2 may be transformed by associating the stereotype <<Dependant>> to all classes at level M1stereotyped <<MT1>>. Figure 10(b) presents graphically how this dependency would be visualized at level M1.

5. Conclusions and related work

The objective of this article is threefold:

- It presents a two-tiered approach to extend UML-M, which consists in creating a heavyweight extension of UML-M and then transforming it into a UML profile. This approach benefits from the expressiveness and readability of heavyweight extensions and of the full standardization of lightweight ones. The extended metamodel is used for defining, maintaining and reasoning about the metamodel, while the transformed UML profile for model definition and portability purposes.
- It defines a procedure to perfom an additive restriction-extension of UML-M in such a way that the semantics of UML-M is preserved. UML-M may be extended, not only with new classes and attributes but also with any MOF-model metaclasses and metaassociations (including associations, dependencies, etc.).
- It shows a methodology to transform an extended UML metamodel into a UML profile, which describes how to transform several metaelements (including associations and dependencies).

We have presented some examples of use of our approach, which show that it may be appropriate for tailoring UML to different domains. In particular, it has been applied [RF00, RF01] to the definition of the metamodel of PROMENADE (a process modelling language in the field of software process modelling [FR99]).

In the last few years, several metamodels have been defined using the 4-layer metamodelling architecture. Some of them carried out a heavyweight UML extension to certain domains, like CWM [CWM00]; others defined UML-profiles, like the UML profiles for business process and software process [UML01]. In other cases, the metamodel was defined from scratch, as a direct instance of the MOF model (UPM [UPM00]).

SPEM [SPE01] is a metamodel to describe a software development process (or a family of such processes) that has been adopted as an OMG specification in december 2001. It is an evolution of the UPM (Unified Process Model [UPM00]). The SPEM metamodelling approach is similar to that of PROMENADE, which was already presented in [RF00, RF01]: an explicit extension of the UML metamodel accompanied by its transformation to a UML profile. SPEM does not focus on presenting a methodology to construct a metamodel extension and to transform it into a profile. Therefore, it does not give details on how an additive UML extension should behave or on how to transform several metaelements like dependencies. Moreover, association transformation is quite limited as we have stated in section 4.2.

[SW01] discusses how different kinds of UML metamodel extensions achieve some features (namely, *readability, expressive power, restrictive power, checkability, conformance*). In particular, they study lightweight extensions (achieved by means of *descriptive and restrictive* stereotypes [BGJ99]) and *heavyweight* extensions. With respect to the latter, it introduces the notion of *restrictive metamodel extension* which is based on the modular structuring of metamodels (separating the abstract metaclasses form the instantiable ones) and which aims at improving model checkability. This approach does not bridge the gap between heavyweight and lightweight extensions (i.e., no transformation methodology is presented). Therefore, it is not fully standard. On the other hand, their definition of *controlled* (similar to *additive*) metamodel extension is quite restrictive. For example, only associations that refine UML associations can be defined. Our notion of additive metamodel extension allows the definition of other kinds of associations and also, the addition of metaelements like dependencies to the extended metamodel.

References

[BGJ99] Berner, S. et alt. A Classification of Stereotypes for Object-Oriented Modeling Languages. LNCS, Vol. 1723. Springer-Verlag (1999).

[CWM00] Common Warehouse Metamodel Specification. Proposal to the OMG ADTF RFP. Common Warehouse Metadata Interchange. OMG document ad/2000-01-01. February, 2000.

[FR99] Franch, X.; Ribó, J.M. Using UML for Modelling the Static Part of a Software Process. LNCS, Vol. 1723. Springer-Verlag (1999).

[JB96] Jablonski, S.; Bussler, C.: *Workflow Management. Modeling Concepts, Architecture and Implementation*. International Thomson Computer Press (1996).

[JPL98] Jaccheri, M.L.; Picco, G.P.; Lago, P.: Eliciting Software Process Models with the E3 Language. ACM Transactions on Software Engineering and Methodology 7(4) October, 1998.

[MOF00] Meta Object Facility Specification. (MOF). Version 1.3 OMG document formal/00-04-03. March, 2000.

[Obj02] Objectering/UML profile builder http://www.softeam.fr/us/pobj_pro.htm

[RF00] Ribó J.M; Franch X.: PROMENADE, a PML intended to enhance standardization, expressiveness and modularity in SPM. Research Report LSI-00-34-R, Dept. LSI, Politechnical University of Catalonia (2000).

[RF01] Ribó J.M; Franch X.: Building Expressive and Flexible Process Models using an UML-based approach. LNCS, Vol. 2077. Springer-Verlag (2001).

[SPE01] Software Process Engineering Metamodel Specification (SPEM). OMG adopted specification pct/01-12-06. December, 2001.

[SW01] Sleicher, A.; Westfetchel, B.: Beyond Stereotyping: Modeling Approaches for the UML. In Proceedings of the 34th Hawaii International Conference on System Sciences (2001).

[UML01] Unified Modelling Language (UML) 1.4 specification. OMG document formal/ (formal/2001-09-67). September, 2001.

[UPM00] The Unified Process Model (UPM) OMG document ad/2000-05-05. May, 2000.