A two-tiered Methodology for Metamodel Extension
Applied to UML 1.4
Franch, X. and Ribó, J. M.
Research Report LSI-04-51-R

Departament de Llenguatges i Sistemes Informàtics

UNIVERSITAT POLITÈCNICA DE CATALUNYA

A two-tiered Methodology for Metamodel Extension Applied to UML 1.4

Xavier Franch[1], Josep M. Ribó[2]

[1]Universitat Politècnica de Catalunya (UPC),
c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona (Catalunya, Spain)
fax: +34.93.401.70.14
franch@lsi.upc.es
[2] Universitat de Lleida (UdL)
C. Jaume II 69, 25001 Lleida (Catalunya, Spain)
fax: +34.973.70.27.02
josepma@eps.udl.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 suffer from several expressiveness limitations. In this report 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 obtain a conservative extension of the UML 1.4 metamodel (as a particular case of any well-defined MOF-model instance) and another to transform an extended UML 1.4 metamodel into a UML profile. As a result, 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. Motivation

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 arise 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). This architecture is outlined in section 2. The UML metamodel is defined at level M2. The MOF model is the UML meta-metamodel and it is defined at level M3.

Several approaches exist to extend the M2 UML metamodel to meet specific needs, each one having specific features.

In this report, we outline those approaches, we discuss some of their properties and we propose an extension methodology which maximizes the properties of *expressiveness*, *standardization* and *understandability* altogether.

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 (1) the expressiveness and understandability that may be reached by means of an explicit metamodel extension and (2) the standardization provided by the UML profiles. Furthermore, the algorithm (a) to perform a restriction-extension of the UML metamodel may actually, be applied to any other instance of the MOF model.

The methodology that we present in this report has been followed in the definition of the metamodel of a Process Modelling Language called PROMENADE [Rib02] as an extension of the UML metamodel and its transformation to a UML profile.

This work has been carried out using the UML1.4 metamodel and the MOF 1.4 [UML01].

2. The 4-layer metamodelling architecture

We use the 4-layer metamodelling architecture as a framework to define the PROMENADE metamodel. The 4-layer metamodelling architecture has been adopted by OMG and used in the definition of the UML metamodel [UML01], the MOF model [MOF00], UPM [UPM00], SPEM [SPE01], CWM [CWM00], etc.

The four layers of this architecture are structured in the following way (see figure 1):

- Layer M3: Meta-metamodel
- Layer M2: Metamodel
- Layer M1: Model
- Layer M0: User objects

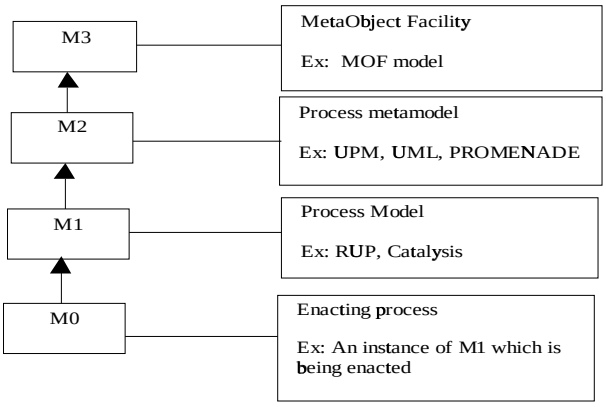

Fig. 1: The four-layer architecture

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.

Metamodelling may be either *strict* or *loose.* In strict metamodelling, each element of a model at level M_j is an instance of exactly one element in the model at level M_{j+1}. However, in loose metamodelling, a level M_j model is an instance of a level M_{j+1} one.

In the following sections we present these four metamodelling layers in a more detailed way.

Layer M3: Meta-metamodel

This layer defines a language to specify metamodels. This language is more abstract and usually more simplified than a specific metamodel based on it (i.e., an instance of it). The natural and standard example of a meta-metamodel is the *MOF model* [MOF00].

The MOF model defines a relatively small set of constructs (essentially, metaclasses and metaassociations) which aim at modelling information (specially from an object-oriented perspective). The importance of the MOF model lies in the fact that it has been adopted as a standard by the OMG and that it has been used as the meta-metamodel for UML.

The MOF model can be used as a domain-independent framework to define the structure and semantics of metamodels in a specific domain (e.g., the UML metamodel, the UPM metamodel, the CWM metamodel, the PROMENADE metamodel...).

The MOF model defines elements like *association, generalizable element, classifier, reference, namespace, data type,* etc. These elements may be instantiated to define specific metamodels.

In addition, the MOF model defines a mapping to the CORBA IDL. This allows the possibility of defining CORBA interfaces for information models which have been described in terms of the MOF model. The client will be able to create, access and update the information described by the model by means of some CORBA objects that represent the metadata.

Layer M2: Metamodel

This layer describes metamodels which represent instances of a meta-metamodel. A metamodel defines a language for specifying a model. Metamodels are usually more elaborated than their respective meta-metamodels and they address certain general domains. Let us present some examples:

- The UML metamodel is an instance of the MOF model addressed to model object-oriented software systems.
- The SPEM [SPE01] is a metamodel for defining software development processes (specially a specific family of related ones, i.e., RUP [JBR99, Kru98], IBM SI Method [LG97], OPEN [GH97], etc.).
- The CWM [CWM00] provides a framework for representing metadata and the processes necessary to manage warehouse data.
- PROMENADE is a language intended to define models for software processes. Hence, the definition of its elements should appear at the level M2 of this architecture, that is, the metamodel level. Since standardization is one of the goals of our approach, we have decided to base the PROMENADE metamodel on the UML metamodel.

The UML metamodel is one of the above-mentioned examples of metamodels (it is the metamodel on which PROMENADE is based). Although it is an instance of the MOF model, this does not mean that every single element of the UML metamodel is an instance of an element of the MOF model. Hence, it is a case of loose metamodelling.

The core of the UML metamodel defines most of the metaelements in the MOF model (some of them as instances). However, this definition is usually richer and more detailed (i.e., not so abstract). For instance, the class *Parameter* incorporates to its definition the type to which it must conform. The class *AssociationEnd* adds several attributes to its definition (changeability, ordering, targetScope…) and some associations (*qualifier, specification, type*).

The UML metamodel defines also many more elements that are not a part of the MOF model (e.g., *StateMachine, SignalEvent, Transition,* etc.). These elements are specific to the domain of object-oriented systems modelling to which UML is addressed. The UML metamodel is described in all detail in [UML01].

Layer M1: Model

This layer describes models. Each model represents an instance of a metamodel. Models provide a language to describe a specific information domain (e.g., the conceptual model for a specific software system, the model for the software system itself, a model for a specific software or workflow process, etc.). The instances of model elements represent objects within that information domain. Examples of model elements may be *Employee* (in the conceptual model of a specific company), *ImplementationDocument* (in a particular SPM), etc.

Usually, the metamodelling relationship between M1 and M2 is strict (i.e., each M1 element is an instance of an M2 metaelement).

Layer M0: User objects (user data)

This layer describes concrete user objects as instances of M1 classes. For instance, the employee *H7867654* or the implementation document *impd123h*. If the model in the M1 layer is a process model, its instances in the M0 layer are objects that belong to the enacting process.

An exemple

- *An element of the M3 layer:* the MOF model defines the meta-metaclass *Class* with an atribute *isSingleton*. An M2 instance of *Class* will set the value of this attribute to true if only one M1 instance is allowed for that M2 element.
- *An element of the M2 layer:* The PROMENADE metamodel, which is an instance of the MOF model in the general domain of Software Process Models (SPM), defines the metaclass *MetaTask* as an instance of the M3 element *Class*. The definition of *MetaTask* sets its *isSingleton* attribute to false so that many instances of *MetaTask* may be defined in the level M1. *MetaTask* defines features as *parameter* or *subtasks* that must be given values when an instance of *MetaTask* is defined.
- *An element of the M1 layer:* A specific SPM (namely, *ComponentLibrary*) is defined in PROMENADE (i.e. *ComponentLibrary* is an instance of the PROMENADE metamodel). *ComponentLibrary* defines specific tasks as *ImplementComponent*, which will be an instance of the metaclass *MetaTask*. Therefore, some specific parameters (*specificationDoc*, *implementationDoc*) and subtasks (*implementOperation*, *setBehaviour*, etc.) must be given in order to conform with the *MetaTask* definition. *ImplementComponent* is an element defined at M1 layer.
- *An element of the M0 layer:* A specific instance of the task *ImplementComponent* is defined. A specific value for each one of the *specificationDoc* and *implementationDoc* parameters and implementOperation and setBehaviour subtasks will be given.

3. Strategies for extending the UML metamodel

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; understandability; 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 understandability altogether, while maintaining robustness and conformance.

3.1 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. This was the initial approach taken by UPM [UPM00] in the context of software process modelling. The following remarks can be made in relation to the above mentioned features.

- *Expressiveness*. This is clearly the most powerful approach to construct a metamodel, since conformity with the UML metamodel, which clearly restricts element definition, 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.

- *Understandability*. 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 metamodelling guarantees a full conformity to the 4-layer metamodelling architecture; loose metamodelling provides model-granularity conformance but not element granularity conformance.

3.2 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.
- *Understandability*. 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 model-granularity conformance.

3.3 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 some of these limitations in the following:

(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 understandability.

(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 figure 2, 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.

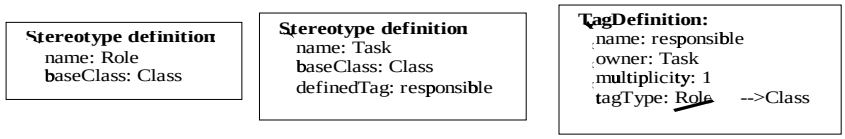

Fig. 2: Association of a *TagDefinition* to a stereotype

(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 is 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 .
- *Understandability*. 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 to the UML metamodel 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. On the other hand, as we have argued before, stereotypes do not have the same semantics as metaclasses.

3.4 Our proposal

Summing up the previous approaches, we may conclude that UML derivation approach provides a high degree of expressiveness and understandability. However, the approach is not standard. UML profiles have complementary features: an optimal standardization but poorer ranks in the other features (see table 1).

Table 1: Summary of the extension approaches

Approaches	Expressiveness	Standardization	Understandability	Robustness	Conformance
MOF instantiation	very high	poor	high	not applicable	both model-granularity and element-granularity conformance possible
UML derivation (heavyweight)	high	fair (problems with extensions)	high	fair	only model-granularity conformance possible
UML profile (lightweight)	poor	high	poor	poor	poor

In this report we propose a methodology based on the combination of heavyweight and lightweight extensions. By 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 figure 3):

1. *Restriction*. Those metaelements which are not of interest in the actual modelling context are discarded from the UML metamodel. We call the result *restricted metamodel*. We remark that this step is not offered by heavyweight extensions; as a consequence, although the metamodel is intended to be tailored to a specific domain, it in fact, includes elements which are not necessary and that make its understandability difficult.
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. This step must not affect the semantics of the remaining UML metaelements 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].

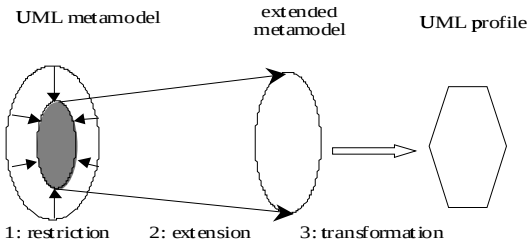

Fig. 3: A two-tiered extension of the UML metamodel

The main advantage of our two-tiered approach is that we use the appropriate metamodel in the adequate context, that is: the extended metamodel to first define 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 5) applied to the extended metamodel, guarantees full compatibility with standard UML.
- *Understandability*. The extended metamodel is represented completely at level M2. This provides a good degree of understandability, 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 with the 4-layer metamodelling architecture. The UML profile to which the extended metamodel has been transformed is not semantically equivalent to it (we have already argued that stereotypes do not have the same semantics as real metaclasses).

In particular, the expressiveness, understandability and conformance drawbacks we have detected in profiles (see section 3) are not as relevant as working directly with profiles because the relevant metamodel with respect to these criteria is the extended metamodel.

The price to pay in our approach is the extra effort needed to transform the extended metamodel into a UML profile. However, it could be argued that even the direct construction of a UML profile as in 3.3 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 5.

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 can a UML metamodel 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 these questions in the next few sections.

4 Algorithm for restriction-extension of a MOF model instance

In this section we present a methodology to perform a restriction and then an extension of the UML metamodel. Furthermore, it may be applied to any MOF model instance (not only the UML metamodel). Given an instance of the MOF model, *m* (level M2), by the application of this methodology, we obtain a metamodel (level M2) *mre* which is an extension of a restriction *mr* of *m*, with the following features:

1. Both *mr* and *mre* are strict instances of the MOF model.
2. *mre* is obtained by instantiation of any class or association of the MOF model (i.e., not only does the methodology allow the creation of new metaclasses and metaassociaations, but also the creation of new dependencies, imports, etc.).
3. *mr* is a conservative restriction of *m*. This means that any element of *mr* retains the same semantics as in *m*.
4. *mre* is an additive extension of *mr.* This means that the semantics of all the *mr* elements is not affected by any means when *mre* is built and, hence, new elements are added to *mr.*

4.1 Why to be (UML-) conservative?

In a conservative restriction-extension *mre* of an instance of the MOF model *m,* each element of *mre* that comes from *m* keeps the same semantics as in *m*. This property is often very useful; in particular, it is very useful in the restrictions-extensions of the UML metamodel (which is the case of PROMENADE), for the following reasons:

- *Standardization.* The conformance to UML semantics enforces standardization, which is probably the main reason to use UML.
- *Well-defined semantics.* If we keep the original semantics of UML in the restriction of the UML metamodel (*UML-M*, from now on), we can rely on the semantics of UML, which is in process of formalization (see section 6). Otherwise, we are likely to introduce ambiguities or

even inconsistencies to the metamodel. Furthermore, the use of the UML metamodel terminology with different semantics may be highly confusing.

- *Portability.* A restriction-extension of the UML metamodel, *mre*, may be transformed into a UML profile, so that instances of *mre* (that is, M1 level models) are portable and can be built using UML modelling tools. The UML-profile will keep the semantics of the elements of *mre* that come from UML only if the restriction *mr* on which *mre* has been built retains the semantics of the UML metamodel.

Usually, restrictions-extensions to the *UML-M* are not conservative. Consider the example of SPEM [SPE01]. SPEM is a MOF-based metamodel which is defined as an extension of a subset of UML (called *SPEM foundation*). However, SPEM cannot be considered as a conservative restriction-extension of the UML metamodel:

(a) *Restriction step.* The absence of some UML elements in the *SPEM foundation* affects the remaining ones, which do not keep the UML semantics (i.e., some elements of the SPEM foundation have not the same structure, associations, ancestors and, hence, semantics as in the UML metamodel).

- Example 1: A UML metamodel *Association*, as a *GeneralizableElement*, inherits several attributes (e.g., *isRoot*, *isLeaf*). A SPEM *Association* does not inherit them (since it is not a *GeneralizableElement*). A UML tool may require a value for those attributes in *Association* instances.

- Example 2: *Package* is a metaeelement from the UML metamodel that has been retained in SPEM. The UML metamodel defines several constraints on *Package* regarding the association i*mportedElement*. Since this association is not kept in SPEM, those constraints make no sense. Furthermore, the transformation of SPEM into a UML profile

(b) *Extension step:* Some added elements may affect the semantics of the UML metamodel ones.

- Example: The addition of the associations *subject-presentation* or *annotatedElement-guidance* which have the UML metamodel element *ModelElement* as one association-end affects the semantics of *ModelElement.*

Summing up, this lack of conformance with UML compromises standardization. In particular, the SPEM foundation elements cannot rely on the semantics of the UML metamodel. As a result, the semantics of the SPEM UML-profile are not well-defined and the portability of SPEM models is impaired.

4.2 Restriction of an instance of the MOF model

A restriction of an instance of the MOF model ***m*** is an instance of the MOF model that (a) contains a subset of *m*'s elements and (b) keeps the semantics of these elements.

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) All elements in *mr* must have the same semantics as in *m*.

This has two implications.

1. Let *a* be any *m*'s MOF association instance (e.g., *depends-on*, *generalizes*, etc.) such that *a* establishes a link from an *mr* element *y* to another element *x* (e.g., *y depends-on x*; *y is a child of x*). In this situation, both *x* and *a* should come up at *mr*. Notice that the direction of the navigation is $y \swarrow x$.

2. Let *a* be any *m*'s association (i.e., *a* is an instance of the MOF class *Association*) that links an *mr* element *y* with another element *x*. If *a* is navigable from *y* to *x*, then both *a* and *x* should come up at *mr*.

Condition (2.1) has several implications: (a) Any *mr*'s MOF class instance which is not the root, must have its supertype in *mr*. (b) Any *mr*'s MOF class instance, must have both its container (if any) and its contained elements (if any) in *mr*. (c) Those constraints associated to a *mr*'s element must belong to *mr*. Moreover, the element to which a *mr*'s constraint is associated, must belong to *mr*. (d) Those types associated to the *mr*'s instances of *TypeAlias*, *Parameter*, *Constant* and *AssociationEnd* must belong to *mr*. (e) Those MOF class instances on which *mr* classes depends, must belong to *mr*. (f) Those namespaces imported by a *mr*'s package (by means of an *Import* object) must be part of *mr*. (g) Those exceptions launched by *mr*'s operations must be defined in *mr*. (h) Those association-ends (and hence, those associations) referred to (and exposed by) *mr*'s references must belong to *mr*. (i) Those model elements to which *mr*'s tags are associated, must belong to *mr*.

On the other hand, condition (2.2) is a bit restrictive. Consider the metamodel (level M2) *m* depicted in figure 4. We want to build a restriction *mr* of *m*. We want that *mr* keeps metaclasses *A* and *C*. For the purposes of *mr*, we consider that it is not necessary to retain metaclasses *B*, *E* and *F*. However, condition (2.2) forces us to keep *B* and *E* due to the existence of the associations *a2* and *a3*. Let us show why condition (2.2) is necessary. We will study the differences between associations *a1*, *a2* and *a3*.

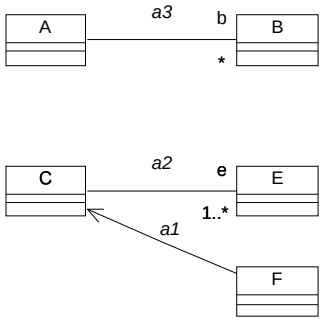

Fig. 4: Restrictions and associations

- *Association a1*: It is not navigable from *C* to *F*. This means that *C* may not make any reference to *F* by means of this association. In particular, no constraint defined on *C* may use this association in order to navigate to *F*. Therefore, *mr* may discard both *F* and *a1*. Notice that condition (2.2) does not force to retain either of them.

- *Association a2:* It is a navigable association from *C* to *E*. Furthermore, its multiplicity on *E*'s side is *1..n*. This means that at least one instance of *E* should be linked to one instance of *C* at level *M1*. Needless to say that *C* may contain references to *E* and that there may be some constraints defined on *C* that use this association to navigate to *E*. It is clear that if we would not keep both *a2* and *E* in *mr*, the semantics of *C* would be modified and we would get a non-conservative restriction of *m*. As a result, condition (2.2) requires that *mr* keeps both *a2* and *E*.

- *Association a3:* It is a similar situation to *a2*. However, in this case, the multiplicity of the association at *B*'s side is * (which includes 0). As before, *A* may contain references to *B* and constraints concerning *B*; therefore, condition (2.2) requires to retain both *a3* and *B*. However, since the multiplicity of *a3* at *B*'s side is *, we may add to any *A*'s subclass (*S*) that we incorporate to *mr* at the extension step (see 4.3) the following constraint: *self.b->size=0*, which forces that no instance of *B* or *B*'s subclass is linked to an *S*'s instance. Furthermore, if we restrict ourselves to instantiate (at level M1) only the metaclasses that we have added at the extension step, we can be sure that no reference to *B* will be present at the model. We will mark this feature by colouring *B* grey. Suming up: the extended metamodel will contain *B* to keep *m*'s semantics but the *B* metaclass will not be used at all, which will be noted by colouring it grey.

As a consequence of the previous discussion, we may consider three kinds of metaclasses regarding the restriction *mr* of a metamodel *m*:

1. *Retained metaclasses:* These are the metaclasses from *m* that represent elements that are necessary in *mr*. Therefore, we will keep them.
2. *Removed metaclasses:* These are the metaclasses from *m* that correspond to elements which are not necessary in *mr* and that are not necessary in a conservative restriction of *m* (i.e., its removal does not change the semantics of the metaclasses that are kept in *mr*).
3. *Unused metaclasses:* It refers to those metaclasses from *m* corresponding to elements that are not to be used in *mr*, but which have associations with other *mr* metaclasses. Unused metaclasses are to be retained in *mr* so that it results in a conservative restriction of *m*.

   If all the associations that link an unused metaclass *c1* and a metaclass *c2* belonging to *mr* have multiplicity * (or 0..1) at the *c1* end, then *c1* will be coloured grey in the *mr* diagrams. Moreover, a constraint may be added to any *c2* child which is defined in the extension step stating that *self.c1->size=0*.

Next we present a procedure to generate a restriction of a MOF model instance (in particular, this instance may be the UML metamodel). This procedure takes into account the above-mentioned considerations.

Restriction procedure

Let *m* be an instance of the MOF model. We want to generate a *m's* restriction called *mr*.
mr should contain the set of elements *E* of *m* that we want to reuse in *mr*. According to the definition of restriction, it should also incorporate transitively the elements linked to those in *E* by means of either instances of the MOF model associations (*DependsOn, Generalizes,* etc.) or instances of the MOF metaclass *Association.* Specifically, *mr* is defined in the following way:

(1) Let $E_0 = E$.

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

- $E_k \subseteq E_{k+1}$
- E_{k+1} includes all the elements linked to an element *e* from E_k 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*. $\Re$ is oriented from *e* to *x*.

  That is, E_{k+1} contains all the elements *x* linked to *e* such that (a) *x* is an *e* supertype; (b) *x* is imported by *e;* (c) *x* is contained in *e;* (d) *x* contains *e;* (e) *x* depends on *e;* *x* is associated to the tag *e* (provided that *e* is a tag); (f) *x* is an association to which a reference defined in *e* refers; (g) *x* is an exception launched by *e;* (h) *x* is used as a type by some *e*, which is a constant, parameter, association end, reference or attribute; (i) *x* is a constraint associated to *e; (j) x* is the element to which the constraint *e* refers (provided that *e* is a constraint).
- E_{k+1} includes all the associations *a* and all the elements *x* from *m* such that *a* is defined between an element *e* from E_k and *x* and *a* is navigable from *e* to *x*.

(3) $mr = \min E_k: k \geq 0: E_k = E_{k+1}$

4.3 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 metametaelement (either a class or an association) defined in the MOF model [MOF00]. Thus, we provide a strict metamodelling extension approach. Specifically, table 2 shows the metaelements that can be incorporated into a MOF model instance in order to extend it.

Table 2: Metaelements that may constitute a MOF model instance

Classes	Class, Package, DataType, Association, Operation, MofAttribute, MofException, Reference, Constraint, Tag, Constant, TypeAlias, Parameter, Import
Associations	Generalizes, DependsOn, AttachesTo, IsOfType, CanRaise, RefersTo, Exposes, Aliases, Contains, Constrains

Notice that this table includes not only classes, attributes and constraints but also dependencies, associations, etc., which are not defined in UML profiles. Therefore, the explicit extension of the UML metamodel is more expressive than the definition of a profile. 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. (see [MOF00]).

We will extend an instance of the MOF model by adding instances of the above-mentioned 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 source 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) A *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 5 shows an example in the context of building a metamodel for SPM. We extend the UML metamodel with the metaclass *SPMetamod* (whose instances are SPMs) and we want to state that the static part of a SPM may consist, among other things, of several (UML) generalizations between some of its constituents (i.e., to create taxonomies of documents or hierarchies of activities). This may be modelled as in figure 5. If navigation from *Generalization* to *SPMetamod* 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 *SPMetamod* 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*.

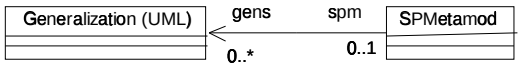

Fig. 5: Extending the UML metamodel with an association

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 an *m*'s element, this element may be subclassified and, hence, some constraints may be defined for this subclass.

Lastly, notice that by enforcing that all elements that are incorporated into the metamodel are instances of a particular MOF-model metaelement, we propose a strict metamodelling.

Extension algorithm

In this section we present an algorithm to generate an extension *mre* of a restricted model *mr*, which is an instance of the MOF model. In order to be more specific and to illustrate our own work in the definition of PROMENADE, the algorithm describes how to obtain an extension of the UML metamodel (*UML-M*) or of a particular restriction of it. However, keep in mind that this algorithm may be applied to the extension of any instance of the MOF model.

In order to generate an extension *mre* from the *UML-M* or from a restriction of it we carry out the following activities:

1. *Identify the set of all the metaclasses that should be contained in mre which either have no correspondence in UML-M or specialize significatively some element of 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 PROMENADE. Instances of the new metaclass *MetaTask*, which represent activities carried out during a software process enactment, may have parameters, which represent documents. Hence, we decide to reuse the *Parameter* UML metaclass and we define an association between *MetaTask* and *Parameter* as in figure 6. 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.

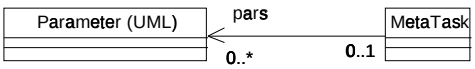

Fig. 6: An inconsistent reuse of the *Parameter* metaclass

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*. Some new packages may be defined which may contain elements from *newels*. Some imports may be associated to *newels;* in particular, *UML-M* elements may be imported by *newels* (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 4.3, 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 4.4). We will see in section 5 that the reuse of *UML-M* associations facilitates the transformation of *mre* 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* or a *DataType*.
   - 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 5.
   - A list of constraints associated to *c*.

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

   In particular, constraints may be associated to any *newel*. However, no new constraint may be applied to a *UML-M* element.

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 *mr*, we may start it over.

4.4 Example

In this section we outline an extension example, which is detailed in [Rib02]: the incorporation of *precedence relationships* (*precedences*, for short) to the UML metamodel.

Precedences
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, TESI].

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.

In [Rib02] it is shown that the concept of precedence is conceptually different from that of UML transition. 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.* Therefore, a precedence will be modelled as a 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.

Figure 7 shows an extension of the UML metamodel which defines several families of precedences which lead 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. Dynamic precedences may be defined precisely only at enactment time.

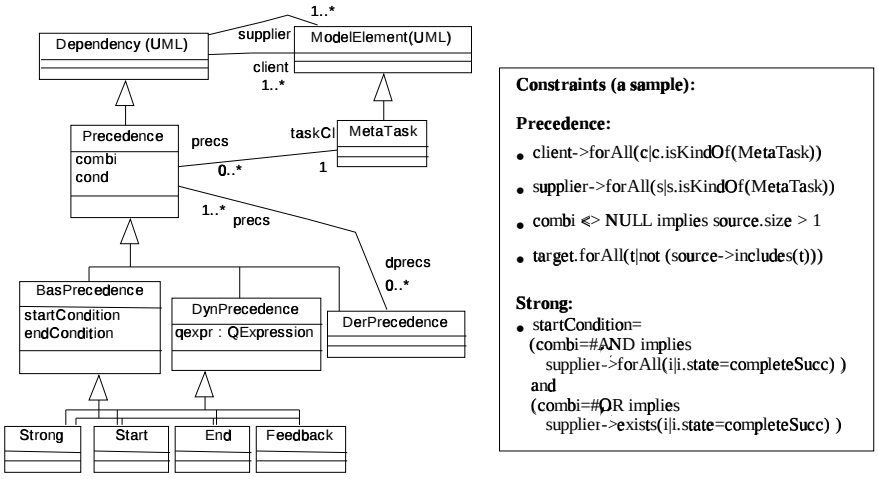

Fig. 7: UML metamodel extended to deal with precedences (fragment)

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.

This example will be addressed in more detail in [Rib02].

5 Transformation to a UML-profile

In this section we present a methodology to transform a metamodel extension constructed following the procedure presented in section 4 into a UML profile. We focus on the most used metaelements to be transformed (namely, class, generalization, attribute, association and dependency).

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 into 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.

5.1 Class and generalization

Each new class *c* that has been incorporated into the restricted-extended metamodel must be transformed in top-down order into a new stereotype *s* such that:

1. The base class of *s* is the closest *c*'s ancestor that belongs to *UML-M*. Notice that, by construction, we can assure that *c* will not have any descendant that belongs to *UML-M*.
2. The name of *s* is the same name as that of *c*.
3. If *c* has some superclasses $\{c_{sup1}, ..., c_{supn}\}$ that do not belong to *UML-M*, the stereotypes $\{s_{sup1}, ..., s_{supn}\}$ into which $\{c_{sup1}, ..., c_{supn}\}$ have been previously transformed, will be *s*'s superclasses (recall that hierarchies of stereotypes are allowed by UML).
4. Add to the stereotype *s* the constraints which are necessary to delimit the semantics of *c.*

   The metaclass *Precedence* from the previous example would be transformed into the stereotype <<*Precedence*>> with base class *Dependency*. The constraints shown in figure 7 would be associated to this stereotype. Appendix A develops this example.

5.2 Attribute

Let *a: T* (where *T* is a data type) be an attribute contained into a class *c* from the restricted-extended model such that *c* does not belong to the UML metamodel. Transform *a: T* into a tag definition *td* such that:

- *td.tagType=T*
- *td.name=a*
- *td.multiplicity*=1
- *td* will have an associated tagged value with *referenceValue.size*=0.

If the attribute type is a collection (*a: collection(T)*), the multiplicity of the associated tag definition will be '*'.

5.3 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 with this approach, the first one methodological and the second one semantic:

- As we have argued in section 4, 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 8). We consider that both association-ends are not navigable.

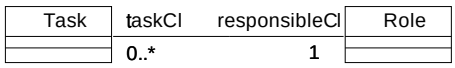

Fig. 8: The "responsibility" association in the extended metamodel

This association may be modelled in the context of a UML-profile as a specific stereotype (<<*ResponsibilityAss*>>) with base class *Association*. <<*ResponsibilityAss*>> represents a special kind of association defined between task classes and role classes. Figure 9 contains the definition of the stereotype while figure 10 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**

p p

Fig. 9: Definition of the <<*ResponsibilityAss*>> stereotype

Fig. 10: An instance of a <<*ResponsibilityAss*>> association

It is important to notice that the explicit extension of the UML metamodel with a new association between a pair of metaclasses is not equivalent to the definition of a constrained stereotype on the base class *Association*. When a metamodel (M2 level) is extended explicitly with a new association, the instances of this association are links stated between classes at model level (M1). For example, the new M2-level association: *role classes are responsible for task classes* induces the M1-level association instance: *the role class TestEngineer is responsible for the task class TestComponent*. In particular, the multiplicity conditions apply to classes at level M1: one task class (e.g., *TestComponent*) has exactly one responsible role class (e.g. *TestEngineer*).

On the other hand, if a stereotype is defined on the UML metaclass *Association* (level M2), the instances induced by this association are links stated between objects at level M0. If a <<*ResponsibilityAss*>> association is represented between *TestEngineer* and *TestComponent*, this induces association instances of the sort: the test engineer *ann* is responsible for testing the components *hashTable* and *catalogue* (in this case, the multiplicity conditions apply to objects at level M0).

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 <<*MetaTask*>> and <<*MetaRole*>>, respectively (constraints 2 and 3); and that only one M1 <<*ResponsibilityAss*>> association may be established having a particular metatask 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.

5.4 Dependency

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 11 contains the definition of the stereotype <<*Dependant*>>.

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

Fig. 11: Definition of the <<*Dependant*>> stereotype

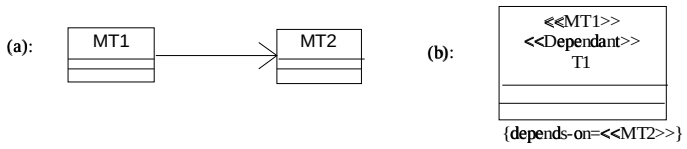

Fig. 12: Example of a M2 dependency and its transformation

Consider a dependency from the metaclass MT1 to MT2 appearing in the restricted-extended metamodel (see figure 12(a)). From section 4 we can infer that MT1 will be a new metaclass that has been incorporated into the extended metamodel and that MT2 may be either a UML metaclass or a newly created one. In any case, a stereotype will have been created for 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 M1 stereotyped **<<**MT1**>>**. Figure 12 (b) presents graphically how this dependency would be visualized at level M1.

5.5 Features of the transformation

It is important to remark again that the the UML profile obtained by the application of the presented methodology on a UML extended metamodel *m* does not keep the semantics of *m*.

As we have already said, the UML extension mechanisms do not provide the same semantics as the usual metaelements. We have presented a two-tiered methodology that allows us (a) to have a well-defined extended metamodel *mre,* which conforms with the 4-level metamodel architecture and which is consistent with UML; and (b) to have a full-standard UML-profile definition *pro,* which result of the transformation of the former.

While *mre* will be used for defining and maintaining the extension, *pro* will be used for portable model definition (instantiation) in any UML-compliant modelling tool.

6 Conclusions and future work

The objective of this report 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 understandability 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). This transformation is not semantic-conservative.

The methodologies that have been presented here has been used in [Rib02] in order to define a metamodel for PROMENADE as an extension of *UML-M* and to transform this extension into a UML profile.

The OMG is about to finish the specification of a significantly new version of UML and MOF, namely, UML 2.0 and MOF 2.0, respectively. These new versions, incorporate important improvements concerning metamodel definition and extension (e.g., modular definition of both metamodels, definition of UML 2.0 as a strict instance of MOF 2.0, sharing of the *InfrastructureLibrary* as a common package of both metamodels, new profile features, etc.). These changes motivate a revision of this work. The preliminar consequences of the adoption of UML 2.0/MOF 2.0 regarding the metodology presented here are the following:

- The concept of two-tiered methodology is even more natural with the new specification, since UML 2.0 has been designed as a strict instance of MOF 2.0 and heavyweight extensions have been explicitly recognized in the UML 2.0 specification as a proper way to extend the metamodel (see [UML04], pp. 23 and 164).
- The statement of constraints to enforce that the extended metamodel does not modify the old one is implictly encouraged, or, at least, suggested by the UML 2.0 specification (see [UML04], p. 164). Hence, it makes perfectly sense to define what a conservative metamodel extension is.
- UML 2.0 keeps and improves the notion of profiles, which constitute the second tier of our extension metodology.
- The restriction part of our metodology may not be necessary in the context of UML 2.0. Its inherently modular structure makes it easier to take some modules as the part of the metamodel that we are interested in extending. This would make the extension process easier.

Currently, we are in process of adapting the metodology that we have described in this report to the context of UML 2.0. As a second step, we plan to define several extension patterns that cover usual situations that modelers come across when they try to extend a metamodel.

References

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

[GH97] Ian Graham, Brian Henderson-Sellers, and HoumanYounessi, The OPEN Process Specification Addison Wesley, London, UK, (1997).

[JB96] Jablonski, S.; Bussler, C.: *Workflow Management. Modeling Concepts, Architecture and Implementation*. ISBN 1-85032-222-8 International Thomson Computer Press (1996).

[JBR99] Jacobson, I.; Booch, G.; Rumbaugh, J.: The Unified Software Development Process. ISBN 0-201-57169-2. Addison-Wesley (1999).

[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.

[Kru98] Kruchten, P: The Rational Unified Process. An Introduction. Addison-Wesley, 1998.

[LG97] Livesey, D.; Guinane, T.: Developing Object-Oriented Software: An Experience Based Approach. ISBN: 0-13-737248-5. IBM Books, Prentice Hall, 1997.

[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

[Rib02] Ribó, J.M.: *PROMENADE: a UML-based Apprach to Software Process Modelling*. PhD thesis. Dept. LSI, Universitat Politècnica de Catalunya, November 2002.

[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.

[UML04] Document ptc/03-09-15 (UML 2.0 Infrastructure Final Adopted Specifcation). OMG document. September, 2003.

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

Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya

Research Reports - 2004

- LSI-04-1-R : *Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations*, Rodríguez, E. and Kapur, D.
- LSI-04-2-R : *Comparison of Methods to Predict Ozone Concentration* , Orozco, J.
- LSI-04-3-R : *Towards the definition of a taxonomy for the cots product´s market* , Ayala, Claudia P.
- LSI-04-4-R : *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-5-R : *Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds*, Willmott, S.
- LSI-04-6-R : *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
- LSI-04-7-R : *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
- LSI-04-8-R : *Studying Embedded Human EEG Dynamics Using Generative Topographic Mapping*, Vellido, A. and El-Deredy, W. and Lisboa, P.J.G.
- LSI-04-9-R : *Similarity and Dissimilarity Concepts in Machine Learning*, Orozco, J.
- LSI-04-10-R : *A Framework for the Definition of Metrics for Actor-Dependency Models*, Quer, C. and Grau, G. and Franch, X.
- LSI-04-11-R : *QM: A Tool for Building Software Quality Models*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-12-R : *COSTUME: A Method for Building Quality Models for Composite COTS-based Software Systems*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-13-R : *Enabling Collaboration in Virtual Reality Navigators*, Theoktisto, V. and Fairén, M. and Navazo, I.
- LSI-04-14-R : *DesCOTS: A Software System for Selecting COTS Components*, Carvallo, J.P. and Franch, X. and Grau, G. and Quer, C.
- LSI-04-15-R : *Evaluation and symmetrisation of alignments obtained with the Giza++ software*, Lambert, P. and Castell, N.
- LSI-04-16-R : *A note on the use of topology extensions for provoking instability in communication networks*, Blesa, M.J.
- LSI-04-17-R : *An ISO/IEC-compliant Quality Model for ER Diagrams*, Costal, D. and Franch, X.
- LSI-04-18-R : *A Case Study on Pruning General Ontologies for the Development of Conceptual Schemas* , Conesa, J.
- LSI-04-19-R : *Adding Efficient and Reliable Access Paths to the JCF*, Marco, J. and Franch, X.

- LSI-04-20-R : *Exploiting Simple Corporate Memory in Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
- LSI-04-21-R : *On the Semantics of Operation Contracts in Conceptual Modeling* , Queralt, A. and Teniente, E.
- LSI-04-22-R : *Complexity issues on bounded restrictive H-coloring*, Díaz, J. and Serna, M. and Thilikos, D.M.
- LSI-04-23-R : *Chromatic number in random scaled sector graphs*, Díaz, J. and Sanwalani, V. and Serna, M. and Spirakis, P.
- LSI-04-24-R : *Bounds on the bisection width for random d-regular graphs*, Díaz, J. and Serna, M. and Wormald, N.C.
- LSI-04-25-R : *Open Source environment to define constraints in route planning for GIS-T*, Pérez, L. and Silveira, A. da M.
- LSI-04-26-R : *A basic repository of operations for the refinement of general ontologies*, de Palol, X.
- LSI-04-27-R : *Tetrahedral mesh subdivision based on underlying volume data*, Rodríguez, L. and Navazo, I. and Vinacua, A.
- LSI-04-28-R : *The Price of Connectedness in Expansions*, Fomin, F.V. and Fraigniaud, P. and Thilikos, D.M.
- LSI-04-29-R : *Smaller kernels for hitting set problems of constant arity*, Nishimura, N. and Ragde, P. and Thilikos, D.M.
- LSI-04-30-R : *Searching Spatial Sense in the Ontological World: Discovering Spatial Objects*, Morocho, V and Pérez, L. and Saltor, F.
- LSI-04-32-R : *Implementation considerations of an Expert System to assess Stream Water Quality management*, Cabanillas, D. and Willmott, S.
- LSI-04-33-R : *Multisided patches*, Pla, N. and Vigo, M. and Cotrina, J.
- LSI-04-34-R : *SVMTool: A general POS tagger generator based on Support Vector Machines*, Giménez, J. and Màrquez, Ll.
- LSI-04-35-R : *A distributed and mobile component system based on the ambient calculus*, Mylonakis, N. and Orejas, F.
- LSI-04-36-R : *Developing Competitive HMM PoS Taggers Using Small Training Corpora*, Padró, M. and Padró, Ll.
- LSI-04-37-R : *The AlignmentSet Toolkit*, Lambert, P.
- LSI-04-38-R : *Integración de Fuentes de Datos espaciales: análisis e implementación de una Ontología de términos espaciales: Primera Parte - - Creación de una Ontología*, Ramos, Erik G.
- LSI-04-39-R : *Integración de Fuentes de Datos espaciales: análisis e implementación de una Ontología de términos espaciales: Segunda Parte - - Evaluación de similaridades*, Ramos, Erik G.
- LSI-04-40-R : *Kernels on Structured Domains*, Valentín, L.

- LSI-04-41-R : *Determining the Structural Events that May Violate an Integrity Constraint*, Cabot, J. and Teniente, E.
- LSI-04-42-R : *Review of Statistical Word Alignment Techniques* , Lambert, P.
- LSI-04-43-R : *Algoritmos geneticos en el problema de la solucion deseadaÒptimizacion de parametros*
  , Barreiro, E. and Joan-Arinyo, R. and Luzón, M.V.
- LSI-04-44-R : *Generative Topographic Mapping as a constrained mixture of Student t-distributions: Theoretical developments* , Vellido, A.
- LSI-04-45-R : *Adapting Agent Communication Languages for Web Service to Web Service Communication*, Willmott, S. and Fernández-Peña, F. O. and Mérida-Campos, C. and Con-stantinescu, I.
- LSI-04-49-R : *A brief on constraint solving*, Hoffmann, C.M. and Joan-Arinyo, R.
- LSI-04-50-R : *Missing data imputation through Generative Topographic Mapping as a mixture of t-distributions: Theoretical developments*, Vellido, A.
- LSI-04-51-R : *A two-tiered Methodology for Metamodel Extension Applied to UML 1ì*, Franch, X. and Ribó, J. M.
- LSI-04-52-R : *Virtual reality for prostate gland cryosurgery*, Joan-Arinyo, R.

Hardcopies of reports can be ordered from:

Núria Sanchez
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Campus Nord, Mòdul C6
Jordi Girona Salgado, 1-3
03034 Barcelona,Spain
nurias@lsi.upc.es

See also the Departament WWW pages, http://www.lsi.upc.es/