A UML Extension to Specify Model Refinements

Natalia Correa
Universidad Nacional de La Plata, Facultad de Informática,
LIFIA - Laboratorio de Investigación y Formación en Informática Avanzada
La Plata, Argentina, 1900
nataliac@sol.info.unlp.edu.ar

and

Roxana Giandini
Universidad Nacional de La Plata, Facultad de Informática,
LIFIA - Laboratorio de Investigación y Formación en Informática Avanzada
La Plata, Argentina, 1900
giandini@sol.info.unlp.edu.ar

Abstract

The refinement technique allows us to capture the relationship between specification and implementation in software developments. The precise documentation of the refinement relationship makes it possible to trace the requirements through the refinement steps. Unfortunately, the standard modeling language UML suffers from a lack of notation to specify refinements in a precise way; in particular, compound refinements can be only partially specified, weakening the traceability potential. In this article, we present an extension of UML to express complex model refinements by means of a well defined composition of elementary refinements. Such extension includes an optional notation to specify complex refinements in an accurate and complete way, thus improving the traceability process.

Keywords: Software Engineering, Modelling Languages, Model Refinements, UML Extension.

Resumen

Por medio de los refinamientos podemos capturar la relación que existe entre la especificación y la implementación en un desarrollo de software. Documentar con precisión esta relación facilita el "rastreo" de los requerimientos a través de los pasos de refinamientos. Desafortunadamente, el lenguaje estándar de modelado UML carece de notación para expresar refinamientos en forma precisa; en particular, refinamientos complejos pueden documentarse solo parcialmente, lo cual disminuye la posibilidad de rastreo. En este artículo presentamos una extensión de UML para expresar refinamientos complejos mediante una combinación bien definida de otros refinamientos más elementales. La extensión propuesta incluye una notación para especificar refinamientos complejos de forma precisa y completa, lo cual favorece al proceso de rastreo.

Palabras claves: Ingeniería de Software, Lenguajes de Modelado, Refinamientos de Modelos, Extensión de UML.

1. INTRODUCTION

Abstraction facilitates the understanding of complex systems by dealing with the major issues before getting involved in the detail. Apart from enabling complexity management, the inverse of abstraction, refinement, captures the essential relationship between specification and implementation. The Abstraction/Refinement relationship makes it possible to understand how each business goal relates to each system requirement and how each requirement relates to each facet of the design and ultimately to each line of the code. Documenting the refinement relationship between these layers allows developers to verify whether the code meets its specification or not, to trace the impact of changes in the business goals and execute test assertions written in terms of abstract model's vocabulary by translating them into the concrete model's vocabulary.

Refinement has been studied in many formal notations such as Z [3] and B [10] and in different contexts, but there is still a lack of formal syntactic definitions of refinement in semi-formal languages, such as the UML.
The standard modeling language UML [13] provides an artifact named Abstraction (a kind of Dependency) to explicitly specify abstraction/refinement relationship between UML models and model elements. In the UML metamodel, an Abstraction is a directed relationship from a client (or clients) to a supplier (or suppliers) stating that the client (the refinement) is dependent on the supplier (the abstraction). The Abstraction artifact has a meta-attribute called *mapping* assigned to record the abstraction/implementation mappings. Refinement mappings are an explicit documentation used to prove that a lower-level specification correctly implements a higher-level one. That is to say, it is a function that maps an abstract specification to a concrete specification.
The more formal the mapping is formulated, the more traceable across refinement steps the requirements are.
Some recently published works analyze the basis of the refinement relationship concepts in UML. In [2], the authors define a formal semantics for the refinement relationship and another subset of UML elements; in [18]the refinement relationship is deeply analyzed and compared with the specialization relationship; Hnatkowska et al. in [8] and Liu et al. in [11] present two different approaches for the definition and use of the refinement relationship, whereas in [9] some formal methods for the automatic exploration of the Dependency concept in the context of UML specification are described.; in [6] the issue of use case refinement is analyzed; in [15] a catalogue of UML refinement patterns is presented.
Although the Abstraction artifact allows for the explicit documentation of the Abstraction/Refinement relationship in UML models, an important amount of variations of this relationship remains underspecified by different causes; for example:

- Some cases of refinement remain hidden under other notations
- Complex refinements have a deficient representation in UML.

The former problem was analyzed in [17], where the authors studied some UML artefacts such as generalization, composite association and use case inclusion; which implicitly define Abstraction/Refinement relationship.
In this article, we focus on the second problem in a syntactic way: UML models representing refinements that are compound by more elementary ones lack precision due to the fact that the UML refinement mapping is able to capture only a part of the information involved in a complex refinement.
Therefore, we present an extension of UML to express complex model refinements by means of a well defined composition of elementary refinements. Such extension includes an optional notation to specify complex refinements. in an accurate and complete way, thus improving the traceability process.
The organization of the article is the following one: in section 2 we discuss refinement relations between models, whereas in section 3 we define, step by step, the construction of a UML extension based on the definition of new stereotypes. In section 4 some examples are presented using the extension proposed. Finally, we present conclusions and lines on future work.

2. MODEL REFINEMENT

According to the step by step refinement technique, models are specified at different levels of abstraction. In general, each refinement step involves the refinement of a group of elementary models contained into a more complex model
The refinement relation between models can be graphically denoted in UML through a kind of Dependency, Abstraction, with stereotype <<refine>> between packages[1], as it can be seen in figure 1. However, this notation only specifies the most external refinement (i.e. the refinement between the packages), nothing is said about the more elementary refinements that are taking place inside each package, such as class refinements, use case refinements, etc.
It can be seen that if model M' refines model M, this refinement will be depicted as a refinement relation between packages representing each model, but it will not be clear which particular element of M' refines which particular element of M.

[1] In the UML metamodel, the metaclass denoting Model is subclass of the metaclass denoting Package.

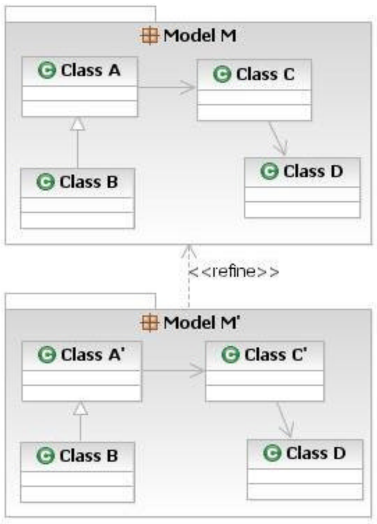

Figure 1. Package refinement. It is not clear which elements of Model M are refined in Model M'

Figure 2 shows notation to specify a refinement between packages M and M' visualizing the refinements between classes that are inside the package. These classes live in different packages that can contain, in turn, other elementary refinements.

This notation may be illegible and very confusing, since for each refining element of the package there must exist an abstraction with <<refine>> stereotype and its corresponding mapping.

The one hand, in order to improve this notation and to make diagrams more readable and clear, a set of stereotypes that specify both where the element that is refined and which mapping associated to that refinement is, are defined. To be able to explicitly express that a class C' of a model M' refines Class C of model M, it is necessary to define elements that denote they are refinements of others. In addition, the application of this notation will be useful if you want to specify that just one element refines some others, specifying the element that are being refined and its associate mapping in each case. In the same way, it could be possible to specify that different elements are, in turn, refinements of some other element.

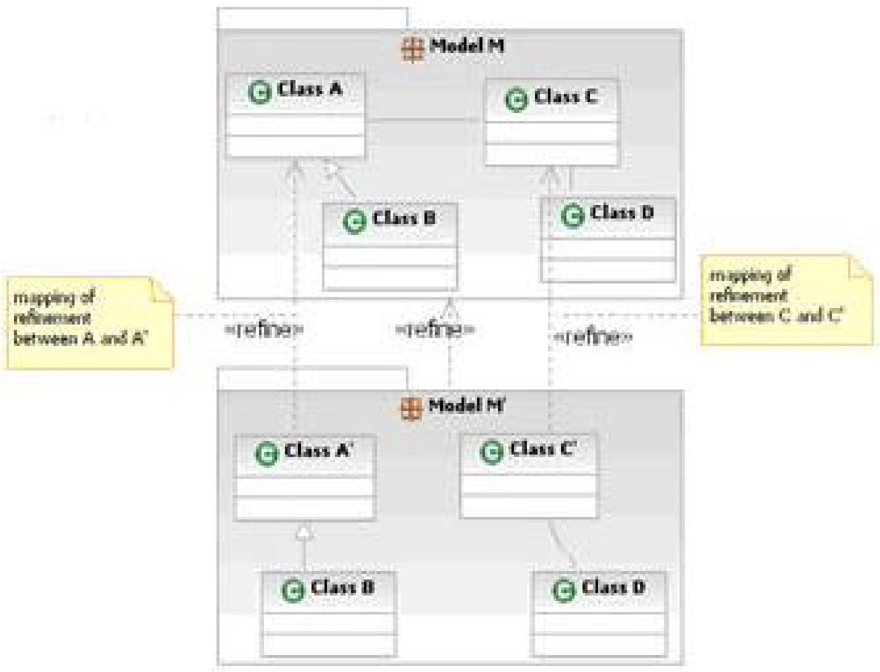

Figure 2. Notation to specify package and its elementary refinements.

On the other hand, figure 2 suggests the existence of a composition between refinements. That is to say, a model refinement is compound of class refinements, use case refinements, etc. This notation is explicitly shown in Figure 3 and it could be used if we rely on a tool that detects compound refinements and allows us to document its components.
Because of the fact that UML does not provide suitable notation to express composition of refinements, in section 3 we present an extension to UML that defines additional modelling elements to overcome this problem.

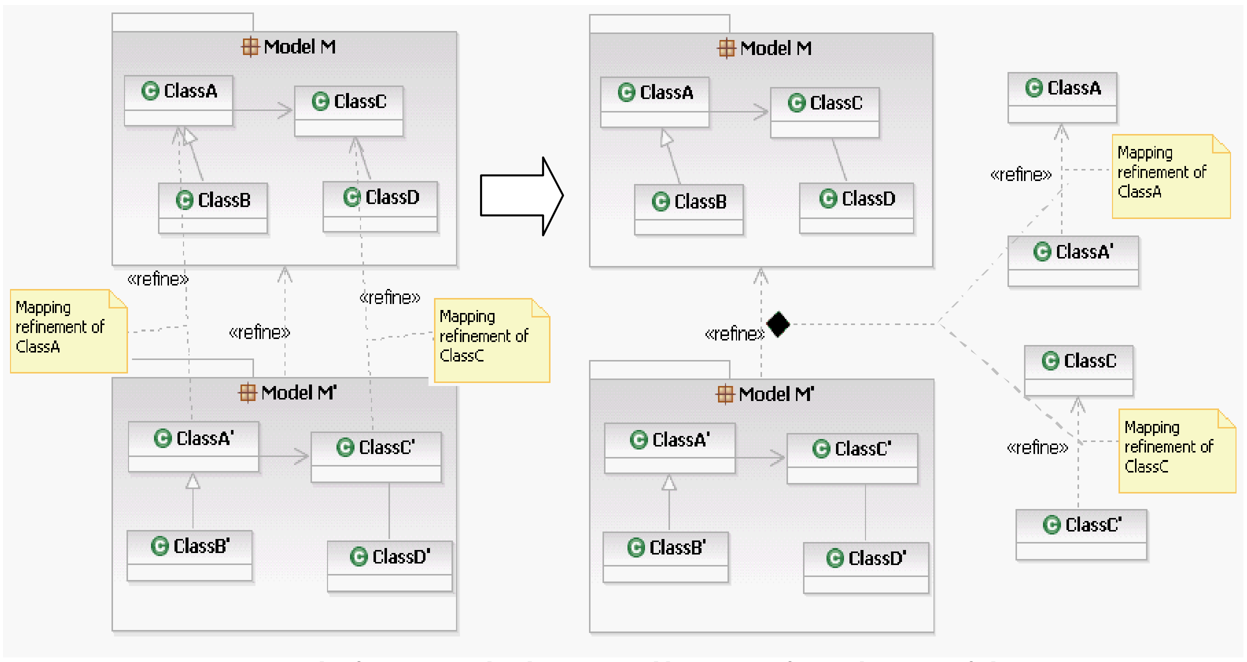

Figure 3. Compound refinement explicitly e xpressed by means of a combination of elementary ones.

3. CONSTRUCTION OF A UML EXTENSION TO EXPRESS MODEL REFINEMENT

Due to the exposed arguments in previous section, it is suggested to extend UML, creating a specific profile that allows us to express particular features of model refinements. A UML profile customizes UML for a specific domain or purpose using extension mechanisms such as stereotypes and constraints. The extension is defined in following subsections in five steps, as it is suggested in other works that define profiles such as [1]; [4]; [7] and [19].

3.1 Definition of the Proposed Metamodel

Initially, the metamodel for the domain that we are studying is defined.
Two levels can be distinguished which are related to each other:
a) Definition of a metaclass to model refined elements (a new metaclass RefinedElement) and, b) modeling both simple refinement relationship (between classes, for example) and compound refinement relationship (between packages). It can be seen in Figure 4.

Each refinement relates two (or more) elements: the abstract and, the refined ones. It can be either a simple or a compound refinement.

a- b-

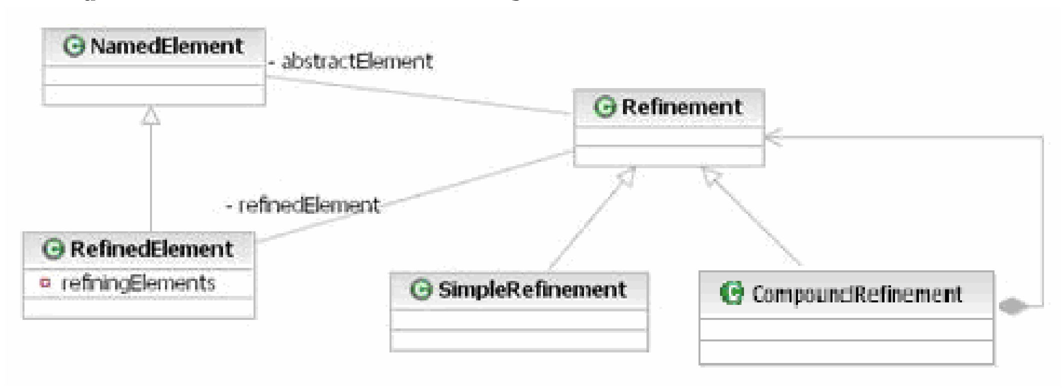

Figure 4. Proposed metamodel

3.2 Definition of Stereotypes Corresponding to the Proposed Metamodel.

A new stereotype for each metaclass and relationship between metaclasses of subsection 3.1, it is defined. They are: one abstract stereotype <<refinement>>, and two more concrete ones, subclasses of <<refinement>>: <<simpleRefinement>> and <<compoundRefinement>>. On the other hand, <<refinedElement>> stereotype is defined. Generally, each one of these stereotypes corresponds one to one with the suggested classes in step 1.
It is not necessary to define a new stereotype for NamedElement, because this abstract metaclass already exists in UML.
In addition, the suggested compound relationship between refinements (pattern "composite" of [5]-see Figure 4), will correspond to a new stereotype <<refinement composition>>.
Figure 5 shows the relation between the suggested metamodel and the defined stereotypes.

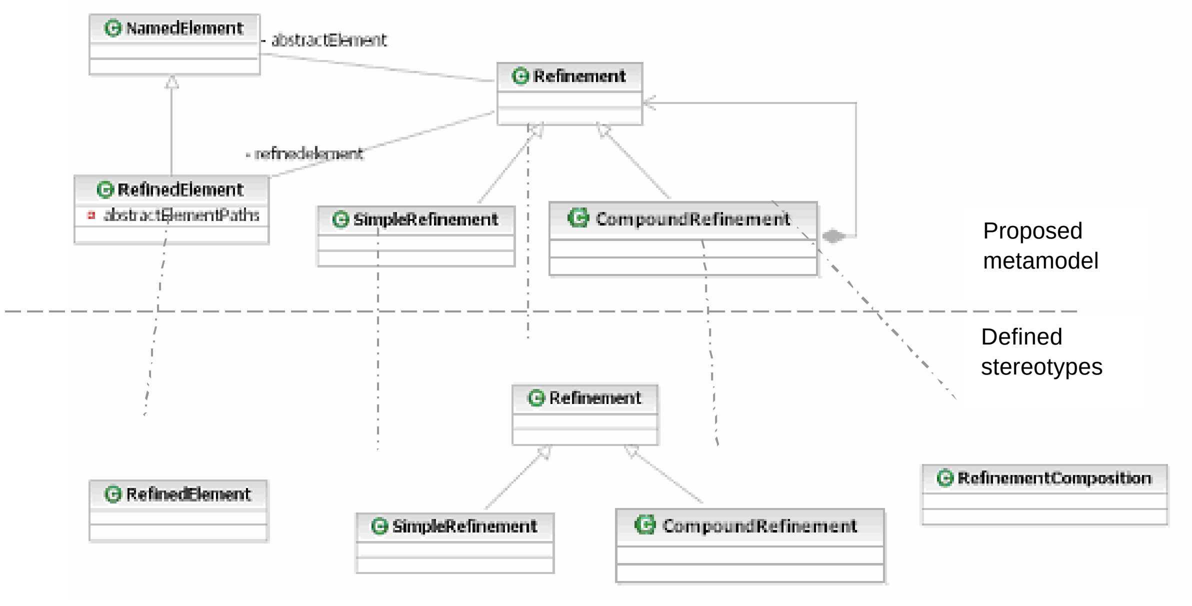

Figure 5. Relation between suggested metamodel and defined stereotypes

3.3 Identification of UML base classes for the proposed stereotypes

UML metaclasses that can be extended to represent the suggested metamodel are identified in this subsection. PackageableElement metaclass is suggested as base of <<refinedElement>>. Initially, RedefinibleElement metaclass was chosen. As redefinable elements can be redefined only in a generalization context, this model element was discarded since refinements may take place in different contexts instead of generalization.
Looking up the metamodel hierarchy, PackageableElement seems to be the required base since refined elements are packageable named elements, as models are.
Stereotype <<refinement>> corresponds to stereotype <<refine>> that already exists in UML and it has a base in Abstraction metaclass (a type of Dependency). Stereotypes <<simpleRefinement>> and <<compoundRefinement>> are <<refinement>> subclass and their base is also Abstraction.
Then, to model composite relationships between refinements, Association metaclass is the most natural choice. As mentioned above, refinements have a base in Abstraction metaclass, which is a kind of Dependency. Since Association relates Classifiers and not Dependencies or Abstractions, Dependency was the chosen metaclass to be the base of <<refinementComposition>>. In figure 6, suggested stereotypes with their base classes can be clearly seen.

3.4 Definition of Attributes for the Stereotypes (Tagged Values).

For each attribute that appears on the suggested metamodel of subsection 3.1, an attribute is defined. These properties (Attributes are Properties in UML [13]) applied to stereotypes are called "tag values".
In case of <<refinedElement>>, it has a "path" to express what element/s (and of which package/s) is/are refined and a "mapping" to specify the way the refinement is made.
As both are related properties, they are denoted as tuples[2] < path, mapping >. Thus, each element that is a refinement of another one (or other ones), will have in its notation a set of tuples. These tuples specify the elements that one element refines.
In section 4, some use examples of the defined stereotypes are presented: a class that refines other ones, one class refined in two classes and refinements compound of more elementary ones.

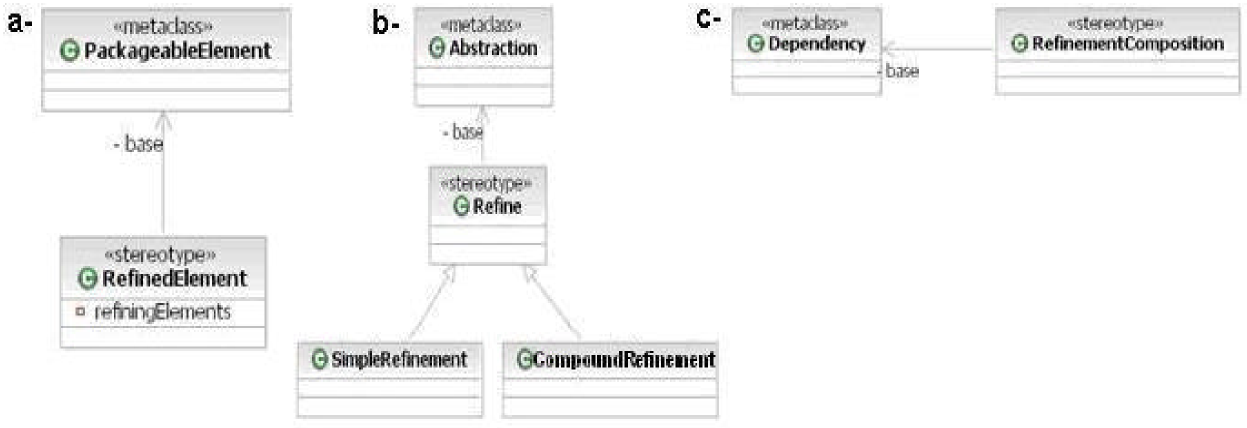

Figure 6. Base classes for proposed stereotypes

3.5 Complete definition of the proposed extension

A formal definition of the new stereotypes previously mentioned is presented, indicating the metaclass that stereotypes (its base) and the relevant constrains, expressed in OCL [14], which must be fulfilled when using them.

- STEREOTYPE refinementComposition

Base: Dependency
Constraints:
1. A << refinementComposition>> relation must be established between refinements (the ends must be stereotyped with stereotypes of the <<refine>> hierarchy)

```
self.source.type.oclIsKindOf(Abstraction) and
self.target.type.oclIsKindOf(Abstraction) and
self.source.type.stereotype.allParents -> includes (<<refine>>) and
self.target.type.stereotype.allParents -> includes (<<refine>>)
```

[2] TupleType exists in OCL. It consists of named parts of same or different types.

2. The end *source* of the relation must be a compound refinement (<<compoundRefinement>>)

```
self.source.type.stereotype = <<compoundRefinement>>
```

3. The end *target* of the relation must belong to the hierarchy of refinements.

```
self.target.type.stereotype.parent -> includes (<<simpleRefinement>>) and
(self.owningPackage.ownedMember -> select (r: Dependency | r.stereotype= <<
refinementComposition >> and r <> self ))-> forAll (r |
r.target <> self.target)
```

- STEREOTYPE compoundRefinement , subclass of refine stereotype

Base: Abstraction
Constraints:
1. A <<compoundRefinement>> Abstraction must be established between models (or packages).

```
  self.supplier.oclIsKindOf(Package) and self.client.oclIsKindOf(Package)
```

- STEREOTYPE simpleRefinement , subclass of refine stereotype.

Base: Abstraction
Constraints:
1. A <<*simpleRefinement*>> Abstraction cannot exist in isolation, it must be "part of" only one compound refinement (between packages), that is: it must be the *target* of a <<refinementComposition>> relationship between abstractions and the *end* is a unique abstraction.

```
self.composite -> size() = 1 and self.composite->stereotype= <<compoundRefinement>>
```

where the *composite* operation returns the set of compound refinements that includes the simple refinement which is part of it.
composite: Abstraction -> Set(Abstraction)

```
composite = (self.owningPackage.ownedMember -> select (r: Dependency| r.stereotype=
<<refinementComposition >> and r.target = self )) -> collect (r: Dependency| r.source )
```

- STEREOTYPE refinedElement

Base: PackageableElement
TaggedValues:
Name: refiningElements
Type: Set (RefinementTuple)
where RefinementTuple= TupleType (path: String, mapping: PropertyCallExp [*])

Note: a PropertyCallExp is an OclExpression that is compound of an operation with a source and a set of arguments (both are OclExpression too)

Constraints:
1. An element with stereotype <<refinedElement>> must have its valid and verifiable set of paths. That is to say, by each defined path must exist the element that is refined in the corresponding model

```
self.paths -> forAll (p: String| self.isValid (self.element(p))
```

where,
paths operation returns the set of paths of the packageable element.

```
paths: PackageableElement -> Set (String)
paths = self.stereotype.refiningElements -> collect(t:RefinementTuple | t.path)
```

element operation returns the NamedElement represented by a path, and it is defined in the context of an packageable element

```
element: String -> Feature
```

isValid operation returns true if a NamedElement is valid (it may be reached) from the packageable element.

```
isValid: NamedElement -> Bool
```

```
isValid (ne) = self.member -> includes (ne)
```
2. All paths must be different from each other (Otherwise, the same element is refined in two different ways in the same context)
```
self.paths -> forAll(p1, p2 | self.element (p1) <> self.element (p2))
```

3. For each path, if the element that is refined belongs to another package, there must be an Abstraction with stereotype <<refine>> between the packages where the package that contains the element that is refined is its “supplier" and the other is its "client".
```
self.paths -> forAll (p | self.owningPackage <> element (p).owningPackage    implies
(self.owningPackage).clientDependency -> exists (a: Abstraction | a.supplier  -> includes
(self.element (p).owningPackage)))
```

4. Consistency rules between the types of elements that are refined, for example:

4.1 An element with stereotype <<refinedElement>> of Class type cannot refine to elements of types Association or Package. (a Class cannot refine to an Association or to a Package) and

4.2 An element with stereotype <<refinedElement>> of Package type cannot refine to elements of types Association or Class. (a Package cannot refine to a Class or to a Association)
```
(self.oclIsKindOf (Class) implies
(self.paths -> forAll (p | oclIsKindOf (self.element (p))  <>  Association) and
(self.paths -> forAll (p | oclIsKindOf (self.element (p)) <>  Package))) and
(self.oclIsKindOf (Package)  implies
(self.paths -> forAll (p | oclIsKindOf (self.element (p)) <> Association) and
(self.paths -> forAll (p | oclIsKindOf  (self.element (p)) <>  Class))
```

5. For each feature of the abstract element, there is a function in terms of other features that defines its mapping and allows for traceability between abstract and concrete elements. For each abstract feature there must exist an element defined in the mapping or a concrete feature under the same name.
```
self.stereotype.refiningElements -> forAll ( t | self.element (t.path).feature -> forAll
( f | t.mapping -> exists (exp | exp.source.referredFeature = f) or (self.feature ->
collect ( fc Í fc.name) -> includes (f.name)))))
```

6. For each OclExpression of mapping, the defined term must be an AttrributeCallExp or OperationCallExp. That is to say, it cannot be any other subclass of OclExpression.
```
self.stereotype.refiningElements -> forAll (t | t.mapping -> forAll ( exp |
exp.source.oclOsKindOf (AttrributeCallExp)  or exp.source.oclOsKindOf
(OperationCallExp)))
```

where *referredFeature* is an additional operation that returns the corresponding associated feature to the ocl element.

```
AttributeCallExp:: referredFeature: Feature
referredFeature= self.referredAttribute

OperationCallExp:: referredFeature: Feature
referredFeature= self.referredOperation
```

7. For each OclExpression of mapping, the term that it is defined corresponds to a feature of the abstract element.
```
self.stereotype.refiningElements -> forAll ( t | t.mapping -> forAll ( exp | self.element
(t.path).feature -> includes exp.source.referredFeature)))
```

8. For each OclExpression of mapping, the definition of the term is compound exclusively of features of the abstract element, of the concrete one or they are literal expressions.
```
self.stereotype.refiningElements -> forAll (t | t.mapping -> forAll ( exp |
exp.allArguments
forAll (a | a.oclIsKindOf (LiteralExpresion) or a.referredFeature.owner =self or a.
referredFeature.owner = self.element (t.path))))
```

4. APPLYING THE PROPOSED EXTENSION

In this section, some examples are presented in order to explain the advantages of the profile that we have suggested. It will be shown how refinement notation becomes more readable, precise and complete.

Use of <<refinedElement>> stereotype.

As mentioned before, a refined element is an element which refines, in its definition, (an) other element(s).
Figure 7 shows a refinement between two models: FlightManager and FlightManager'. Some model elements contained by them, class FlightEmployee and class Flight, are involved in this refinement as the refining elements and class FlightEmployee' and class Flight' are the model elements that represent the refined elements. Basically, one of the refining attributes of class FlightEmployee was named workedHours. As a refinement of this class, FlightEmployee' was stereotyped and specified as follow:

- the path denotes the qualified name of the element which *FlightEmployee'* refines. In this example, the path is *FlightManager:: FlightEmployee*;
- the mapping denotes the way the refinement is made. In this example, attribute *workedHours* is defined in terms of the concrete attributes *previousFlightHours* and *flightHours*.

Notice that a refinement relationship with <<refine>> stereotype and its corresponding mapping should be denoted for each inner elementary refinement.

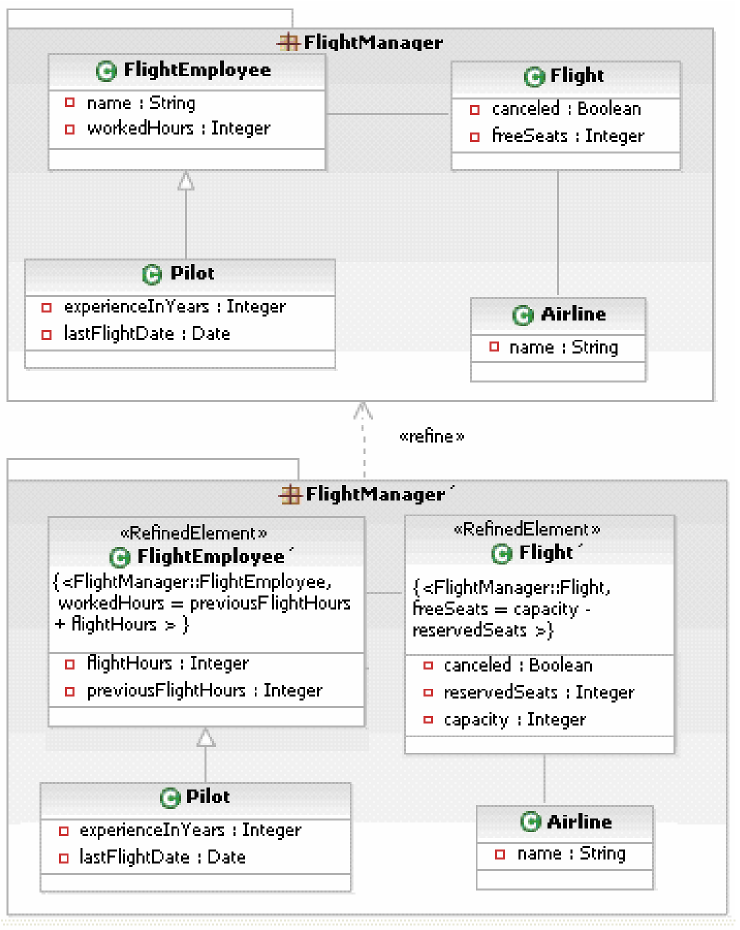

Figure 7. Refinement between models using the suggested extension

Another case of refinement was represented in figures 8 and 9: one element represents the refinement of the other two elements. Figure 8 was introduced to see the graphical notation of these elements without the use of the suggested stereotypes. And then, in Figure 9, the same example with the application of stereotypes is shown.

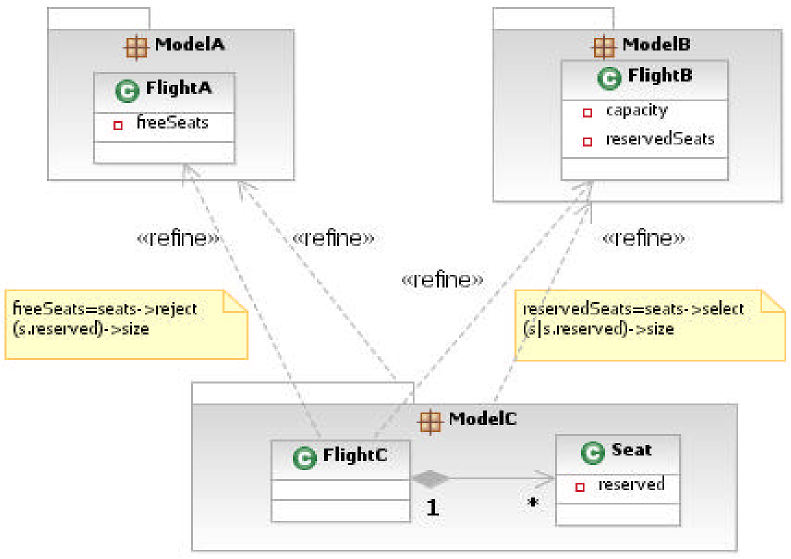

Figure 8. One element represents the refinement of other elements

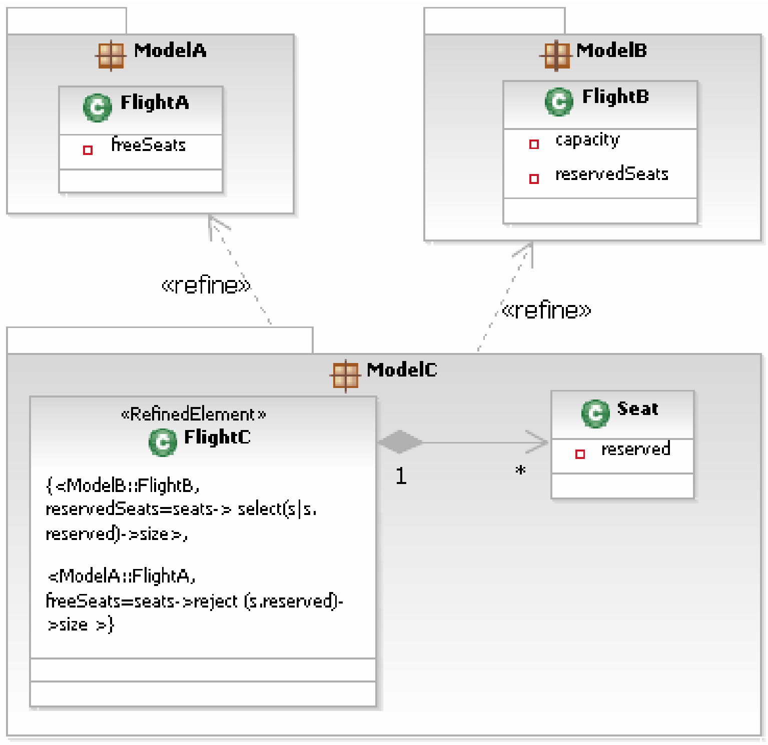
ModelA
FlightA
freeSeats
ModelB
FlightB
capacity
reservedSeats
«refine»
«refine»
ModelC
«RefinedElement»
FlightC
{<ModelB::FlightB, reservedSeats=seats-> select(s|s. reserved)->size>,
<ModelA::FlightA, freeSeats=seats->reject (s.reserved)->size >}
Seat
reserved
1
*

Figure 9. One element represents the refinement of other elements with suggested notation.

Use of <<refinementComposition>> stereotype

Figure 10 shows the full use of the presented stereotypes that allows us to model this example: a way to represent elementary refinements that are hidden under compound refinements, is presented. Notice that the figure explicitly shows the compound refinement relationship. That is to say, a model refinement (between packages FlightManager and FlightManager') is compound of more elementary refinements (between classes FlightEmployee and FligthEmployee', and classes Fligth and Fligth'). So, the first is a compound refinement, and the second ones are simple refinements.

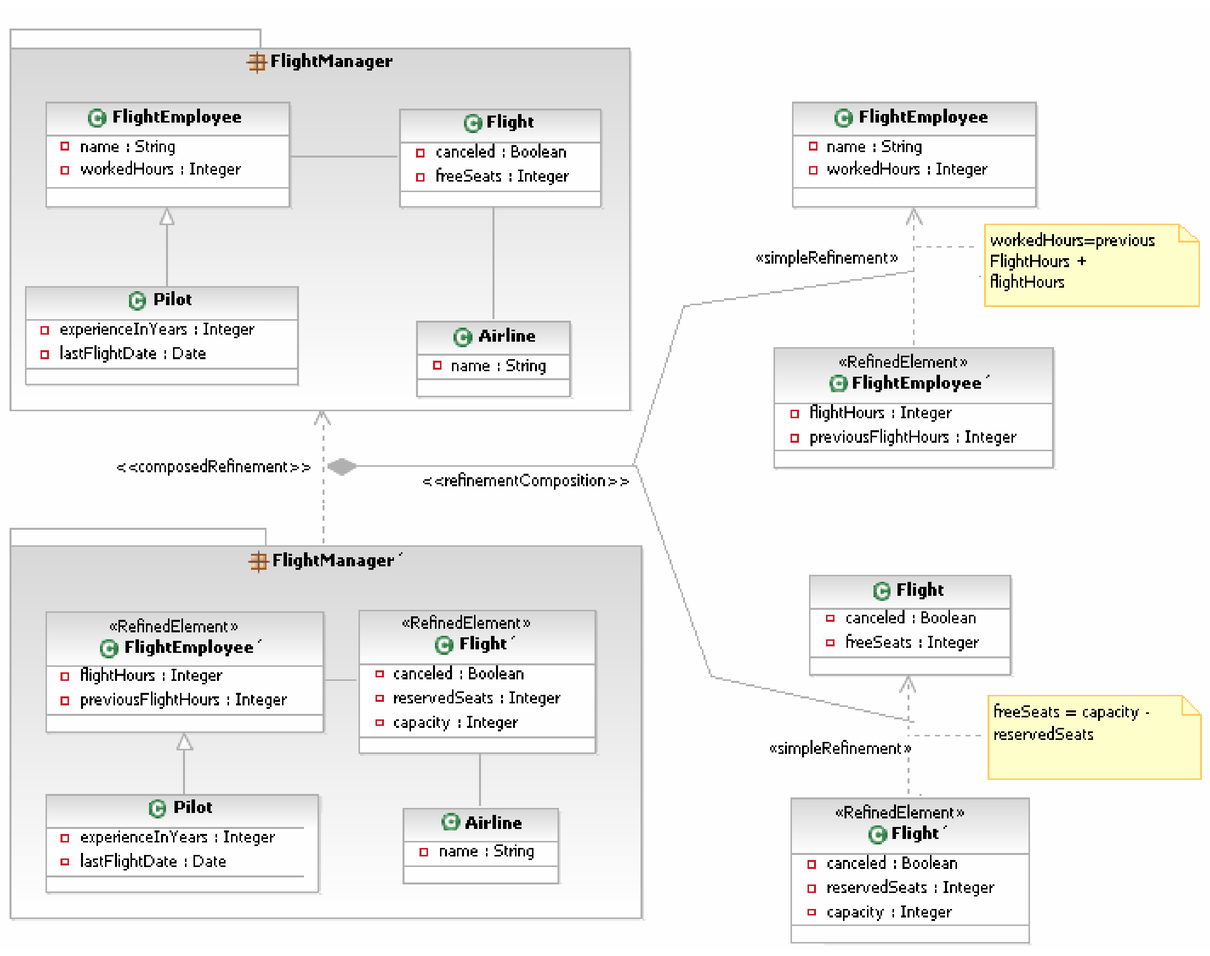

Figure 10. Full use of presented extension.

5. CONCLUSIONS AND FUTURE WORK

The more detailed the documentation of refinements (mapping) in software development is formulated, the more accurately requirements could be traced through the refinement steps.
In the present article, we have focused our attention on the study of composition of refinements. The standard modeling language UML suffers from a lack of notation to specify complex cases of refinement, due to the fact that the UML refinement mapping cannot capture the elementary refinements which form a compound refinement. Package refinement is one of these cases. When a Package refines another one, which is really refined are the elements that live inside the package: classes, cases of use, etc. Consequently, compound refinements remain under specified in UML models, which obstruct the traceability process.
As a solution for this weakness we have presented a detailed construction of an extension for UML to model refinements, based on the definition of new stereotypes.
In addition, very frequently only one element can represent the refinement of several others or vice versa, and possibly these elements do not belong to the same model. Therefore, in the suggested extension, we also included stereotypes that allow us to model these refinements in an optional but more readable way than UML offers. Such stereotypes are being incorporated into Pampero [16], an experimental case tool integrated with Eclipse that was developed in our university.
Furthermore, model refinement can be thought as a particular way of model transformation. Models transformation plays a very important role in MDA[12].
As a line of future work, we will include the possibility of defining a standard language for transformation of models based on the construction of a new profile.

References

[1] Blankenhorn Kai & Jeckle Mario, “A UML Profile for GUI Layout”, NODe 2004, LNCS 3263, pp.110-121, 2004 Springer-Verlag Berlin Heidelberg 2004.

[2] Davies, J., Crichton, C. Concurrency and refinement in the UML. Electronic Notes in Theoretical Computer Science, Volume 70, July 2002

[3] Derrick, J. and Boiten,E. Refinement in Z and Object-Z. Foundation and Advanced Applications. FACIT, Springer, 2001

[4] Fuentes,L.,Troya, J.M., Vallecillo, A.. “Using UML Profile s for Documenting Web -based Application Frameworks”. Annals of Software Engineering, Vol. 13, pp. 249-264, June 2002.

[5] Gamma, E., Helm,R., Johnson,R., and Vlissides,J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley Publis hing Company, 1995.

[6] Giandini, R., Pons, C., Pérez,G. Use Case Refinements in the Object Oriented Software Development Process. Proceedings of CLEI 2002, ISBN 9974-7704-1-6, Uruguay. 2002.

[7] Grassi,V., Mirandola, R., and Sabetta,A. “A UML Profile to Model Mobile Systems”. Seventh International Conference on UML Modeling Languages and Applications, UML 2004. Lisboa, Portugal.

[8] Hnatkowska B., Huzar Z., Tuzinkiewicz L. On Understanding of Refinement Relationship. Workshop in Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships at the 7th International Conference on the UML. Lisbon, Portugal, October 11, 2004.

[9] Kostrzewa M. Exploration and Refinement of the UML Dependency Concepts. Workshop in Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships at the 7th International Conference on the UML. Lisbon, Portugal, October 11, 2004.

[10] Lano,K. The B Language and Method. FACIT. Springer, 1996.

[11] Liu Z., Jifeng H., Li X., Chen Y. Consistency and Refinement of UML Models. Workshop in Consistency Problems in UML-based Software Development III at the 7th International Conference on the UML. Lisbon, Portugal, Oct 11, 2004

[12] OMG. MDA Guide, v1.0.1, omg/03-06-01, June 2003.

[13] OMG. The Unified Modeling Language Specification – Version 2.0, UML Specification, revised by the OMG, http://www.omg.org, April 2004.

[14] OMG. The Object Constrain Language Specification – Version 2.0, for UML 2.0, revised by the OMG, http://www.omg.org, April 2004.

[15] Pons, C. Heuristics on the Definition of UML Refinement Patterns. 32nd Int. Conf. on Current Trends in Theory and Practice of Computer Science. SOFSEM (SOFtware SEMinar). January 21 - 27,2006 . Czech Republic.

[16] Pons C., et all. Precise Assistant for the Modeling Process in an Environment with Refinement Orientation. In "UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers". The tool can be downloaded from http://sol.info.unlp.edu.ar/eclipse

[17] Pons, C., Kutsche, R. Traceability Across Refinement Steps in UML Modeling. 3rd Workshop in Software Model Engineering WiSME at the 7th Int. Conference on the UML. October 11th 2004. Lisbon, Portugal .

[18] Pons, C., Pérez,G., Giandini, R., Kutsche, R. Understanding Refinement and Specialization in the UML. and 2nd International Workshop on MAnaging SPEcialization/Generalization Hierarchies (MASPEGHI). In IEEE ASE 2003, Canada.

[19] Ziadi,T. Hélouët,L., and Jézéquel,J.M. “Towards a UML Profile for Software Product Lines”, PFE 2003, LNCS 3014, pp. 129–139, 2004 Springer-Verlag Berlin Heidelberg 2004