Understanding Refinement and Specialization in the UML

C.Pons[1], G.Perez[1], R.Giandini[1] and R.Kutsche[2]

[1] *LIFIA – Laboratorio de Investigación y Formación en Informática Avanzada*
University of La Plata, Buenos Aires, Argentina - email: cpons@info.unlp.edu.ar
[2] *TU-Berlin, FB Informatik*, Berlin, Germany

ABSTRACT

The old technique of "*abstraction and refinement"* makes it possible to understand complex systems by describing them in successive levels of detail. On the other hand the more modern technique of "*generalization and specialization"* (or Inheritance) facilitates the construction of systems by enabling reuse of specifications. Both techniques enable developers to specify a taxonomic relationship between a more general element and a more specific one.

Abstraction is frequently used as a mere synonym for generalization -respectively refinement as a synonym for specialization. Confusion also stems for the occasional use of the same specification notation for both concepts.

However, these terms have different implications and the lack of distinction is the cause of much wrong model interpretations.

The purpose of this article is to analyze both refinement and specialization relationship between UML model elements, in order to clarify the main differences (and similarities) between them.

Keywords: modeling languages, UML, semantics, formalization, abstraction, refinement, generalization, specialization..

1 Introduction

Modeling is the central issue of analysis and design. A model is a blue print for systems, it describes the structure and behavior of things either as they exist or as it is intended to build them. The model constitutes the fundamental base of information upon which the problem domain experts, the analysts, the software developers and the testers interact. Thus, it is of a fundamental importance that it clearly and accurately expresses the essence of the problem, but this goal is difficult to achieve; models tend to contain errors, omissions and inconsistencies because they are the result of a complex and creative activity .

There is an old technique, named "***abstraction and refinement***" ?Dijkstra, 76? which provides great advantages towards the improvement of model's clarity and understandability (and as a consequence model's accuracy). Abstraction makes it possible to understand complex systems and to deal with the major issues before getting involved in the detail. An abstract model shows information in only as much detail as necessary; a refinement is a more detailed description that conforms to another (its abstractions). Every property specified in the abstract model holds in the refinement too, but possibly more properties hold in the refinement.

Apart from enabling for complexity management, the refinement technique captures the essential relationship between specification and implementation. Development by refinement steps allows one to check whether the code meets its specification or not. This relationship can be traced across refinement steps, from the requirement specification to the code.

On the other hand, there is a more modern technique named "***generalization and specialization***" (or Inheritance) [Booch, 91? which is a central issue in the object oriented paradigm. It is applied to enable reuse, so that less effort is spent when we re-specify things that have already been specified in a more abstract or more general way. In the object oriented paradigm a Class describes the structure and behavior of a set of objects (all the instances of that Class). However it does so incrementally by describing extensions (increments) to previously defined classes (its

parents or superclasses). Incremental development applied to Classes originates a subtype relation between the parent class and the child class. The more specific element is fully consistent with the more general element (it has all of its properties, members, and relationships) and may contain additional information. For example, to say that all Savings are BankAccounts is the same as saying that Savings is a subclass (or subtype) of BankAccounts. Notice that this mechanism can be misused leading to subclasses which are not subtypes, this problem was deeply analyzed in ?Wegner and Zdonik, 88? ?Cook, Hill, and Canning, 94?.

The standard modeling language UML (Unified Modeling Language) [UML-OMG, 2001] provides special notations to specify both relationships described above: generalization/specialization relationship is expressed by means of an artifact named Generalization, while abstraction/refinement relationship is represented by an artifact named Abstraction. Figures 3 and 4 show the UML notation for Generalization and Abstraction, respectively.

Abstraction artifact relates sets of elements that represent the same concept at different level of abstraction or from different point of view. Frequently this relationship is established between model elements at different steps in the development process, such as analysis and design. Notice that abstraction/refinement relationship can be established between either two model elements of the same kind (e.g. an analysis class and a design class) or two model elements of different kind (e.g. a use case model being refined by a collaboration model). On the contrary, Generalization artifact always appears connecting two model elements of the same kind.

There exists strong semantic overlapping between both relationships; both of them specify a taxonomic relationship between a more general element and a more specific one. In practice, this overlapping is reflected in the fact that abstraction is often used as a mere synonym for generalization -respectively refinement as a mere synonym for specialization. However, each one of these terms has different implications and the lack of distinction is the cause of much wrong model interpretations. .

The contribution of this article is to provide insight on the informal dialectic by appealing to both intuition and to formal definitions. By putting these concepts on a solid footing we disambiguate the discourse and provide a foundation for formal reasoning and analysis.

Moreover, the UML defines the concept of GeneralizableElement, which is a model element that may participate in a Generalization relationship. As expected, UML defines that a Class is a GeneralizableElement, but also UML considers that other model element, such as Association, Stereotypes, Collaborations and Use Cases, may be treated as GeneralizableElements. The concept of generalization/specialization hierarchy is well understood when it is applied on Classes and in general it is compatible with the concept of incremental inheritance hierarchy. In other words, any element in a generalization/specialization hierarchy is considered as an increment of its parents. But, if we try to keep this compatibility with inheritance, when the concept of generalization/specialization hierarchy is extended to other model elements several contradictions and ambiguities arise.

In addition we will look for argumentations to discern which kind of UML model elements can (properly) participate in a generalization/specialization hierarchy or in an abstraction/refinement hierarchy, and which is the role each model element plays in each hierarchy.

Our discussion will be organized according to the primary modeling concepts of the OO paradigm offering a simple view of a system as a group of collaborative objects. Therefore, the primary modeling concepts we will consider are:

1- individual objects (structure and behavior of individual objects)

2- relationships between individual objects (associations, aggregations)

3- joint actions (how a group of individual objects collaborates with each other)

Abstraction/refinement and generalization/specialization techniques can be applied on each one of these modeling concepts. Considering that objects, its associations and its collaborations are not independent of each other, the refinement and/or specialization of one concept in general impacts on the others.

2 Object Refinement

Let's analyze the task of creating models of a single kind of object using the refinement technique. Let U be an object universe. Let W be a

part of the universe we are interested in describing (i.e. W? U). The world W can be characterized in different ways or in different abstraction levels, as follows:

Let P^{w}_{1}, ..., P^{w}_{k} be predicates characterizing the world W.

Each characteristic predicate P^{w}_{i} allows one to distinguish the set of objects in W from the rest of individuals in the universe; that is to say x belongs to W if and only if $P^{w}_{i}(x)$ holds. Predicate P^{w}_{i} is constructed using observable characteristics of objects in the universe. For example:

Let U be the set of financial objects, W be the set of bank accounts, $P^{Account}_{1}$, $P^{Account}_{2}$, $P^{Account}_{3}$ be predicates characterizing to W, as follows:

$P^{Account}_{1}(x)$: "x is a statement of money kept at a bank"

$P^{Account}_{2}(x)$: "x is a record of the valuables that a customer deposited in a bank, which can be withdrawn later on"

$P^{Account}_{3}(x)$: " x is a statement of money kept at a bank, identified by a unique number"

A partial order can be defined between the characteristic predicates in the following way:

P^{w}_{i} ? P^{w}_{j} if and only if ($P^{w}_{i}(x)$? $P^{w}_{j}(x)$)

We say that P^{w}_{j} is more abstract that P^{w}_{i} and that P^{w}_{i} is more detailed (or refined) that P^{w}_{j}

For example: $P^{Account}_{3}$? $P^{Account}_{1}$, while no order can be established neither between $P^{Account}_{1}$ and $P^{Account}_{2}$, nor between $P^{Account}_{2}$ and $P^{Account}_{3}$.

Noticed that all the predicates describe the same world, but in different level of detail or from different point of view, that is to say:

W= *instances*(P^{w}_{1}) = ... = *instances*(P^{w}_{k})

Where *instances* is a function returning the set of objects characterized by P^{w}_{i} , that is to say: *instances*(P^{w}_{i}) = {x? U | $P^{w}_{i}(x)$}.

Refinement relationship between sets of objects can be obtained in different ways; the more frequently occurring form of refinement is the incremental refinement consisting in defining an extension (or increment) of a model in order to obtain a more detailed or specific one. If P is a predicate characterizing world W, and P´ has the form (P? Q) being Q any predicate, it is straightforward to prove that P´ ? P, because the formulae (P? Q? P) is a tautology. We have observed two forms of incremental refinement:

Case a. "Homogeneous Refinement".

An homogeneous incremental object refinement occurs when the increment applies to all the individuals in the world. For example $P^{Account}_{3}$ is a homogeneous refinement of $P^{Account}_{1}$. Figure 1 shows on the left hand side a hierarchy of Bank Account specification and on the right hand side a possible set of instances described by them. The top level contains an abstract description of Bank Account, then the bottom level contains a more refined descriptions obtained by adding details to the previous one that apply to all the individuals in the set (i.e. all the accounts in the bank have got a unique identification number).

Case b: "Heterogeneous Refinement".

As a consequence of adding more and more detail to the description, we usually discover a new characteristic that it is not present in all the

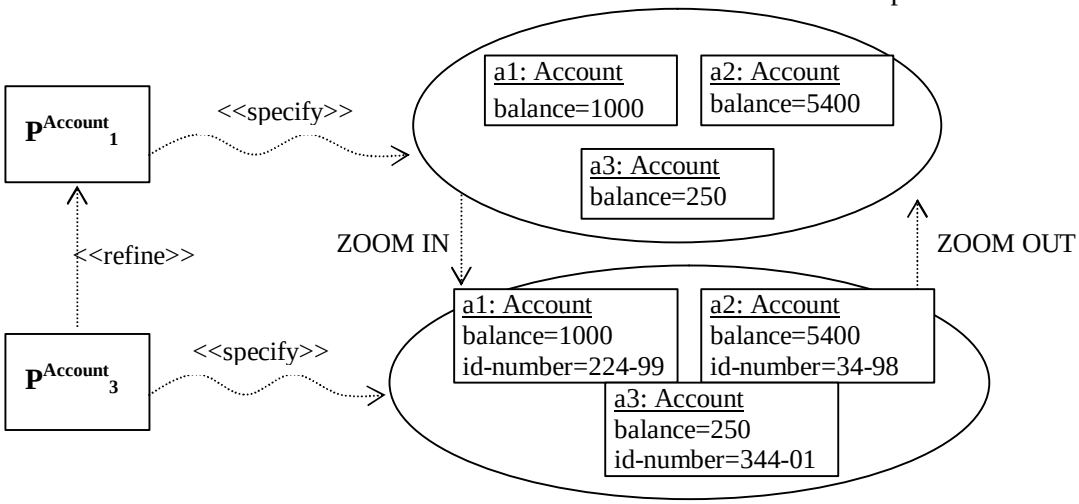

Figure 1: Homogeneous refinement

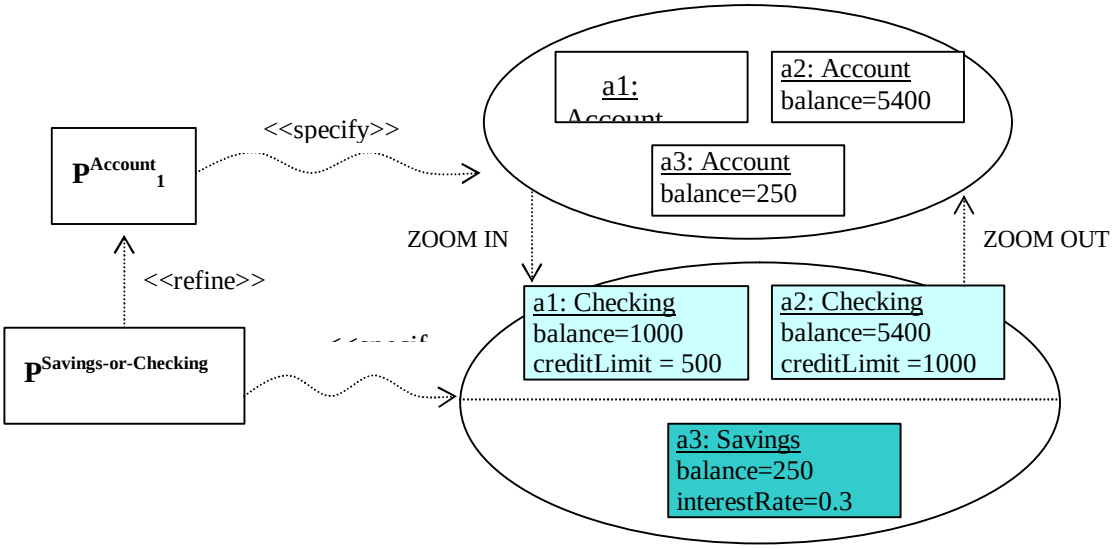

Figure 2: Heterogeneous refinement

individuals in the world, that is to say different subsets have different characteristics. Then world W becomes partitioned into two or more sub-worlds:

$W = W_1 ? ...? W_n$

Characterization of world W is obtained from the characterization of its sub-worlds, as follows:

$? x ? (P^{W}(x) ? P^{W1}(x) ? ... ? P^{Wn}(x))$

That is to say, an individual belongs to W if and only if the individual belongs to some sub-world W_i.

For example: figure 2 shows two kinds of Accounts: Savings Account and Checking Account. Let $P^{Savings}$ and $P^{Checking}$ be the characteristic predicates of each sub-world respectively:

$P^{Savings}(x)$: " x is a statement of money kept by a customer at a bank on which interest is paid"

$P^{Checking}(x)$: " x is a statement of money kept by a customer at a bank that enables the customer to pay by check until a certain limit of credit".

We can see that $P^{Savings-or-Checking}$ is a heterogeneous incremental refinement of $P^{Account}{}_1$, where predicate $P^{Savings-or-Checking}$ is the joint of both $P^{Savings}$ and $P^{Checking}$.

The two forms of incremental refinement described above gives rise to the main difference between refinement and specialization of object's description, as we explain in the following sections.

2.1 Heterogeneous Object Refinement vs. Specialization in the UML

The UML semantics document explains the meaning of the Generalization construct in terms of segment descriptors. A full descriptor is the full description needed to describe an instance. It contains a description of all the attributes, associations, and operations that the instance contains. In OO languages, the description of an instance is built out of incremental segments that are combined using inheritance to produce a full descriptor for an instance. The mechanism of inheritance defines how full descriptors are produced from a set of segments connected by generalization.

In the UML metamodel the prefix ***all*** is used to denote inherited features. For example, the additional operation *allFeatures* can be applied on any Class, resulting in a set containing all Features of the Class itself and all its inherited Features, as follows:

allFeatures =

self.features ->union (self.parents.allFeatures)

Figure 3 shows a UML generalization-specialization hierarchy, where Account is the generalization and Savings is the specialization, the expression Savings.allFeatures evaluates to the set integrated by balance, interestRate, deposit(), withdraw() and payInterest().

In ?D´Souza and Wills, 1998? experts say that the difference between incremental development

(i.e. generalization-specialization hierarchies) and refinement is that the refinement is a self contained model, nor just an extension. But this assertion alone is insufficient to characterize the differences. We think we have to consider two dimensions: syntax and semantics.

The assertion "one model is self-contained while the other is just an extension" only takes into consideration the dimension of syntax, which is not really important because from an incremental development we can derive a self-contained refined model in a straightforward way. For example, when people read the model in figure 3, people think of Savings as being a complete self-contained model, not just an extension, that is to say:

Savings = Account + ?Savings, where ?Savings is the increment specified by the lower class in the hierarchy in Figure 3.

Figure 4 shows the refinement relationship induced by the specialization relationship in Figure 3. Therefore, the example makes evident that in

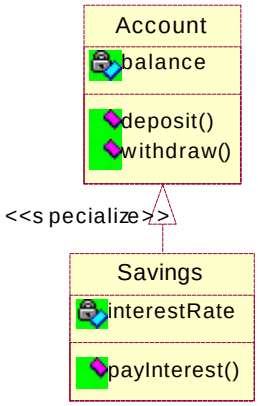

Figure 3: generalization/specialization hierarchy in UML

this dimension the difference between incremental development (i.e. specialization) and refinement is not substantial, it is just a syntactical abbreviation matter, while the intuitive interpretation of both descriptions of Savings (the one in figure 3 and the one in figure 4) remains the same.

Now let's turn our attention to the dimension of semantics, considering the object domain denoted by each specification. Notice that although in figure 4 Savings is a self-contained model, it is not a complete refinement of Account, because it describes just a part of the world of Accounts. A refinement should describe the very same world (eventually in different level of detail). That is to say, we have a "heterogeneous refinement" that generates a partition of the world of Accounts into two sub-worlds: Savings and No-savings. Where No-savings denotes the set of individuals belonging to Account but not to Savings (i.e. No-savings = Account - Savings). The joint (Savings È No-savings) is effectively a complete refinement of Account, as depicted in figure 5 (notice that the notation represents a UML Abstraction artifact with one Supplier (the Account) and two Clients (Savings and No-savings). Notice that Abstraction relationship whit more than one Client is described in the UML metamodel but there is not a standard notation to represent it).

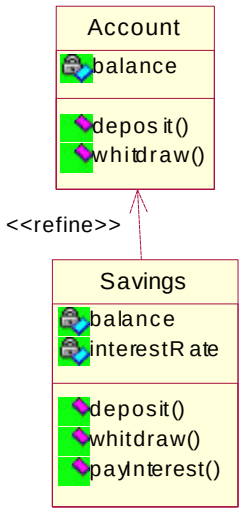

Figure 4: abstraction/refinement in the UML.

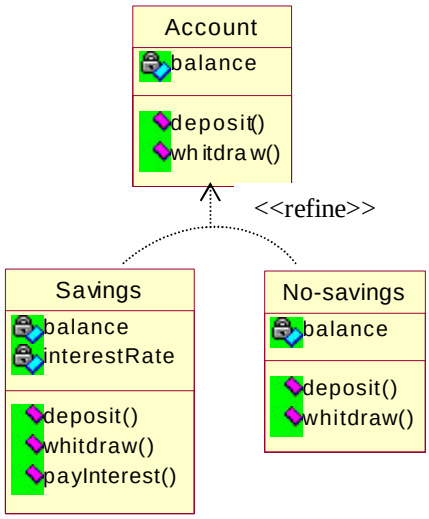

Figure 5: complete refinement of Account

In conclusion, the difference between specialization and refinement techniques encloses two conceptual dimensions: a syntactic one and a semantic one which considers the scope of models. The former is of a little relevance, it is just an abbreviation, while the later is of a fundamental importance to understand the concept of model refinement and to distinguish it from the concept of model specialization. For example, Savings is a specialization of Account, but not a refinement. A refinement should describe the very same world as the one described by its abstraction, nor just a part of it.

In short, Generalization artifact allows modelers to implicitly specify heterogeneous incremental refinements.

2.2 Homogeneous Object Refinement in the UML

In this section we present some examples of frequently occurring forms of homogeneous refinement.

Homogeneous object refinement by composite

One case of homogeneous object refinement occurs when the object is refined revealing its constituent parts. Figure 6 shows an example in the Banking domain, where ManagementDepartment is refined revealing three parts inside: ClientManager, LoanManager and AccountManager.

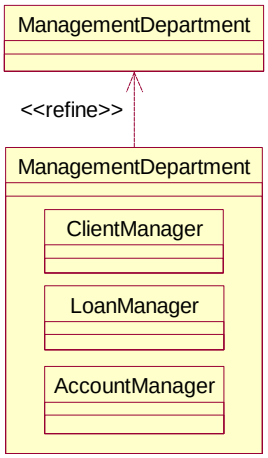

Figure 6: Object homogeneous refinement by composite

Homogeneous object refinement by realization

A homogeneous object refinement by realization is usually used in programming languages, such as Java. In figure 7 the Account class implements the BankAccount interface, i.e. the class is obliged to define all operations specified in the interface.

An interface is a specification for the externally-visible operations of a class, component, or other classifier without specification of internal structure.

Conceptually, the relationship between BankAccount and Account is a homogeneous refinement. In UML, this relationship is represented by an Abstraction relation with stereotype <<realize>>, named Realization.

Graphically, the Realization relationship from a class to an interface that it supports is depicted by a dashed line with a solid triangular arrowhead (a "dashed generalization symbol").

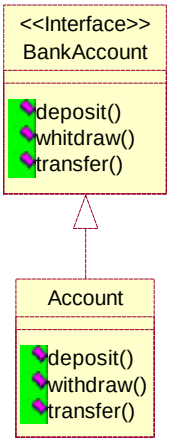

Figure 7: Object homogeneous refinement by realization

This notation could generate confuse interpretations due to the fact that the Realization notation is very similar to the Generalization notation but, semantically, these two relationships represent different concepts, as explain in section 2.1.

3 Association Refinement

During the development process both objects and its relationships are gradually refined. It is usual to zoom in or out in both dimensions at the same time. As soon as you resolve one object into several, you must introduce new relationships

specifically between them. And as soon as you discriminate a set of objects you must discriminate the relationships between them too.

The UML provides an artifact named Association to specify relationships between objects. An Association defines a semantic relationship between Classes. Each instance of an Association is a set of tuples relating instances of the corresponding classes. An Association has at least two Association Ends. Each End has a name and defines a set of properties of the connection (e.g. which Class is connected, multiplicity, navigability).

An instance of an Association is a set of tuples. In the simplest case of binary association, its potential instances are sets of pairs of classifier instances, as follows:

instances: Association -> Set Set(Instance x Instance)

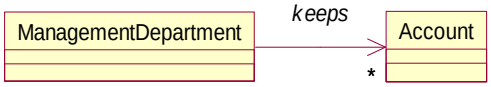

Figure 8: Abstract Association between ManagementDepartment and Account

For example, in figure 8:

instances(*keeps*) ? ? { (x,y) (x ? instances(ManagementDepartment) and y ? instances(Account) }

Where the function named instances applied on a Classifier returns the set of instances of that Classifier.

As well as Classes, Associations are amenable to be both refined and specialized. In a parallel direction with the classification of Object refinement, we define two different kinds of refinement between Associations: homogeneous refinement and heterogeneous refinement, as follows:

Case a: "Homogeneous Refinement".

A homogeneous refinement takes place when an abstract association is described in more detail, for example adding information about multiplicity or visibility, etc. The most interesting case of homogeneous refinement occurs when one or more participants are refined revealing several parts. For example, figure 8 shows an abstract association between Account and ManagementDepartment. Afterwards, as a consequence of applying the homogeneous refinement showed in figure 6, the abstract association between ManagementDepartment and Account has to be refined to show that the Account is related to the AccountManager instead of being related to the ManagementDepartment as a whole. Figure 9 illustrates this situation.

Case b: "Heterogeneous Refinement".

A heterogeneous refinement occurs when an abstract association is described in more detail and as a consequence of adding more detail some differences between the set of links emerge. Abstract association becomes partitioned into two or more sub-associations.

3.1 Heterogeneous Association Refinement

The most usual forms of heterogeneous refinement occur when one or more participants

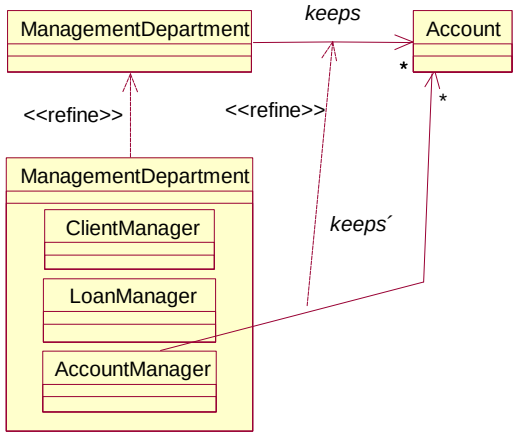

Figure 9: Homogeneous refinement of Association

are refined revealing either a composite or a sub classification:

Case b.1 Heterogeneous refinement by composite

Figure 10 shows an abstract association between ManagementDepartment and Client. When the ManagementDepartment is refined (figure 6), this association is refined by two associations: one between ClientManager and Client and the other between LoanManager and Client This situation is showed in figure 11.

This is a heterogeneous refinement, due to the fact that it splits the initial set of links into two subsets.

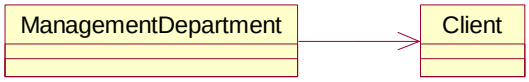

Figure 10: Abstract Association between ManagementDepartment and Client.

Notice that a homogeneous object refinement can originate either a homogeneous or a heterogeneous refinement of the associations connecting the refined objects, as we showed in the two examples above (Figure 9 and Figure 11).

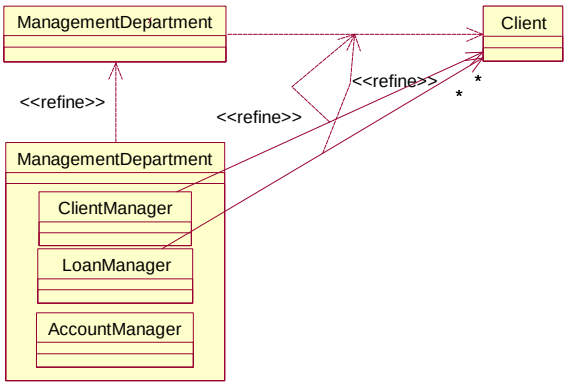

Figure 11: heterogeneous refinement of Association by composite

Case b.2 Heterogeneous refinement by subclassification

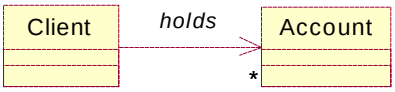

Figure 12: Abstract Association between Client and Account

Figure 12 shows an association between Account and Client. Afterwards a heterogeneous refinement is applied to Account generating two classes: Savings and Checking. At the same time, a heterogeneous refinement is applied to Client and two new classes show up: NormalClient and SpecialClient. In this domain normal clients are only allowed to open Savings while special clients are only allowed to open Checking accounts. Model in figure 13 illustrates this situation**.**

The UML specifies that Association is a GeneralizableElement. According to the definition of inheritance, it is expected that a child association inherits some properties from its parents, but this is not the case: in the context of the Association artifact the only additional operation with prefix all in the UML specification document is allConnections that results in the set of all AssociationEnds of the Association itself. It is defined as follows (see that connections belonging to parents are not considered at all):

allConnections:Association(Set(AssociationEnd)

allConnections = self.connection

Consequently, it becomes clear that the meaning of generalization hierarchy of Associations is different from that of Classes. While generalization hierarchy of classes reflects incremental refinement, generalization hierarchy of association seems to be not incremental at all.

Analyzing the UML model in figure 13, we see that the UML definition of allConnections is reasonable because child Association does not inherit parent's properties (such as parent's connections); child association specializes parent's connections. Let A, B and C be the associations Client-holds-Account, NormalClient-holds-Savings and SpecialClient-holds-Checking respectively:

A.connections={e1,e2}where e1.type= Client and e2.type= Account

B.connections={e3,e4} where e3.type= NormalClient and e4.type=Savings

It makes no sense that Association *B* inherits parent's connection between Account and Client, because it would become a quaternary association which is not the intended meaning of the diagram.

Semantically, the generalization/specialization relationship between associations denotes an inclusion relation between the corresponding sets of instances. That is to say, if B is a specialization of A, then instances(B) ? instances(A).

In the example in figure 13 the association A is partitioned into two sub-associations: B and C. The conjunction of both sub-associations gives rise to a refinement of A:

A = B ? C

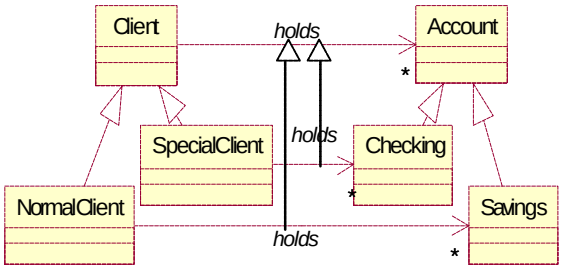

Figure 13: heterogeneous refinement of Association by subclassification

A pair (x,y) belongs to an instance of A if and only if the pair belongs to either an instance of B or an instance of C, that is to say:

instances(A) = instances(B) ? instances(C)

For example, figure 14 illustrates an instantiation of the abstract Association A and its refinements B and C, as follows:

instances(A) = {(juan, a1),(pedro, a2),(pedro, a3) }

instances(B) ={ (juan, a1) }

instances(C) = { (pedro, a2), (pedro, a3) }

Some notes about methodology:

- As it happens with method overriding, child association should have the same name as parent association. For example NormalClient overrides the association *holds which* it inherited from Client. Overriding simplifies the notation because the relationship between both associations becomes obvious, then the Generalization diagram can be removed (figure 15).
- Frequently, we face heterogeneous refinement of Association of the form A ? B ? C where dom(B)? dom(C)?? or ran(B)? ran(C)?? . For example, model in figure 15 specifies that Normal Clients are allowed to hold only Savings Accounts while Special Client can have both Savings and Checking accounts. Class SpecialClient inherits the Association with Account from its parent, so any instance of SpecialClient can be connected to any instance of Account (including subclasses). In this case the refinement of the abstract association *holds* is not explicitly depicted in the model. That is to say, *holds* ? Normal-*holds*-Savings ? Special-*holds*-Account, where association Special-*holds*-Account is implicit in the model, as follows:

Instances(holds) ?

? { (x,y) ? x? instances(Client) ^

y? instances(Account) }

Instances(Normal-*holds*-Savings) ?

? { (x,y) ? x? instances(NormalClient) ^

y? instances(Savings) }

Instances(Special-*holds*-Account) ?

? { (x,y) ? x? instances(SpecialClient) ^

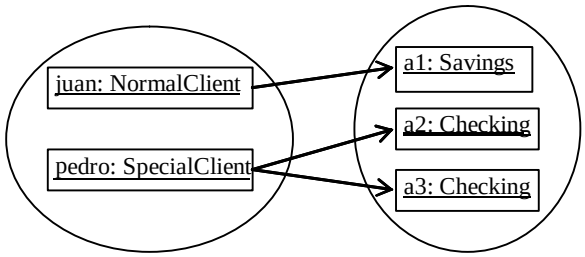

Figure 14: instantiation of association between subclasses of Client and Account.

(y? instances(Savings) ? y? instances(Checking) }

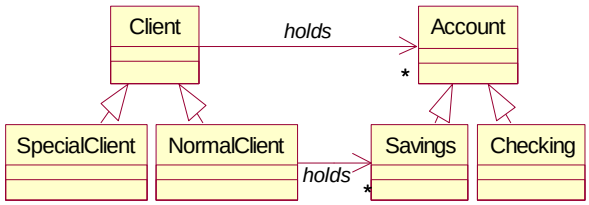

Figure 15: implicit refinement

3.2 Homogeneous Association Refinement

In the case of homogeneous refinement the association is not partitioned in several sub-associations, instead of that each link is "refined". For example, in figure 9 the association established between Account and AccountManager is a refinement of the abstract association established between Account and ManagementDepartment:

instances(*keeps)* ? instances(*keeps´*)

For example, figure 16 illustrates an instantiation of the abstract Association *keeps* and its refinement *keeps´*, as follows:

instances(*keeps)* = { (MDOfBostonBank, a1) (MDOfCitiBank,a2) (MDOfCitiBank,a3) }

instances(*keeps´*)= {(BostonAM, a1) (CitiAM,a2) (CitiAM,a3) }

Notice that each link forming the association is described in more detail. Instead of showing the connection between Account and ManagementDepartment, the refined association shows the connection between the Account and the specific part of the ManagementDepartment that is in charge of it.

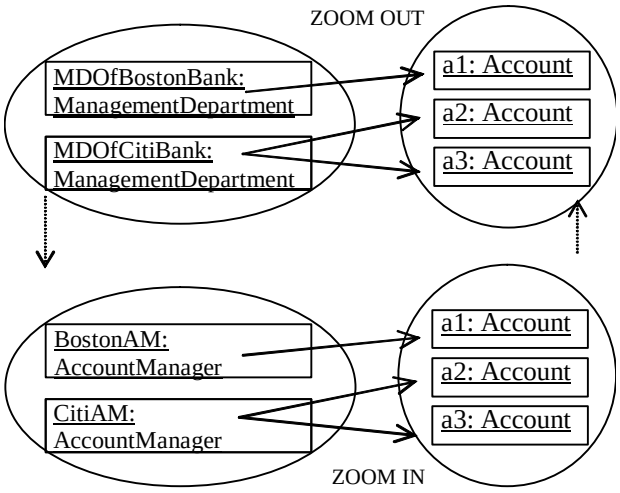

Figure 16: instantiation of association between ManagementDepartment and Account

Another example

This example illustrates another case of homogeneous association refinement. When two interfaces are related by an abstract association, the classes implementing the interfaces must be related between them, too. The later association represents a homogeneous refinement of the former, as depicted in figure 17, where the interfaces BankClient and BankAccount are related by an

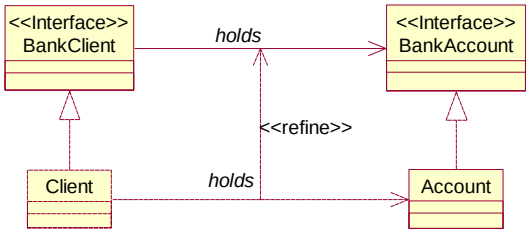

Figure 17: homogeneous refinement of Association

abstract association. Then class Client realizing BankClient must be associated to a class realizing BankAccount, in this case Account.

4 Joint Action Refinement

A joint action is a description of how a group of individual objects collaborate with each other. A joint action is described by a collection of actions that take place between the objects. The UML provides a number of artifacts to specify actions, we restrict our discussion here only to Use Case diagrams.

Action abstraction is the technique of treating an interaction between several participants as one single action. Then it is possible to zoom into, or refine, an action to see more detail. What was one single action is now seen to be composed of several actions. Each one of these actions can be split again into smaller ones, into as much detail as required. Figure 18 shows an abstract use case and its refinement. In the abstract model the action Buy is treated as a single action then the more refined model shows that the Buy action is composed by three sub actions: Pay, Select and Collect. To specify composite actions UML provides a relationship between Use Cases called *Include*.

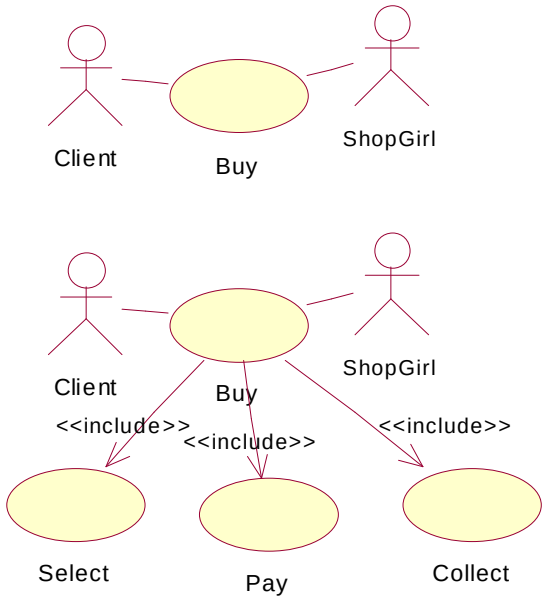

Figure 18: use case refinement

On the other hand, the UML defines that Use Cases are GeneralizableElements, so a use case may specialize a more general one. More exactly, UML specification document says, "a generalization relationship between use cases implies that the child use case contains all the attributes, sequences of behavior and extension points defined in the parent use case, and participates in all the relationships of the parent use

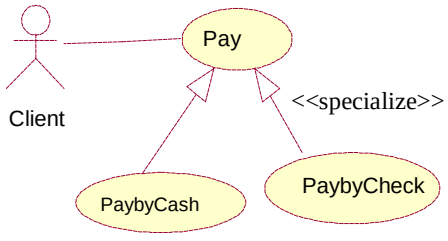

Figure 19: Use Case specialization

case". Figure 19 shows a general use case and its specializations?Cockburn, 2000?. The general use case describes a Payment, while each specialization describes a particular kind of payment: PaybyCash and PaybyCheck.

These two examples illustrate the differences between refinement and specialization of use cases. It is quite evident the correspondence between these two use case relationships and the two forms of refinement discussed before (i.e. homogeneous and heterogeneous refinement). In the first case we have a homogeneous refinement, instances(BUY) ? instances(BUY´), while in the second case we have a heterogeneous refinement, instances(Pay) ? instances(PaybyCash) ? instances(PaybyCheck), where instances of a use case is the set of all allowed traces described by it ?Harel and 2002?.

5 Conclusion

At the present the Unified Modeling Language is considered the standard modeling language for object oriented software development process. The specification of UML constructs and their relationships is semi-formal, i.e. certain parts of it are specified with well-defined languages while other parts are described informally in natural language. There is an important number of theoretical works giving a precise description of core concepts of UML and providing rules for analyzing their properties. See for instance the works of [Evans et al.,1998;1999], [Kim and Carrington, 1999], [Breu et al., 1997], [Övergaard, 1998, 1999], [Pons and Baum, 2000, 2002], [Pons et al., 2000]. But, several UML concepts still need deeper analysis and formalization.

In this article we focus on two UML artifacts - *Abstraction* artifact specifying abstraction/refinement hierarchies and *Generalization* artifact describing generalization/specialization hierarchies - providing a formal basis for the distinction between these terms. This paper is based on our previous works reported in [Pons 2002] [Giandini et al., 2002].

We formulated a mathematical description of these artifacts which makes it evident the existence of a partial overlapping between the semantics of both artifacts: the later corresponds to one particular case of the former, named "heterogeneous refinement".

The analysis reported in this article contributes to the improvement of UML syntax and semantics. Formalization of the UML is an important task because the lack of accuracy in its definition causes wrong model interpretations and discussion regarding the model meaning. The interpretation done by the people who read the model may not coincide with the interpretation of the model creator. These misunderstandings lead to the highly expensive problem of construction of systems that do not meet user expectations. Finally, tools supporting graphical specifications, where intuitive perceptions are insufficient, will benefit by accurately defining this distinction.

Our analysis not only copes with generalization/specialization hierarchies of Classes which is the more frequently occurring form of generalization/specialization hierarchy in OO modeling, but also considers hierarchies of Associations and Use Cases and it can be extended in order to consider the remaining UML generalizable (and refinable) elements too.

References

Booch, G.,Object Oriented Analysis and Design with Applications. Benjamin Cummings, 1991.

Cook, W, Hill, W.and Canning, P Inheritance is not subtyping, In Theoretical Aspects of OO Languages, MIT Press, 1994.

Breu,R., Hinkel,U., Hofmann,C., Klein,C., Paech,B., Rumpe,B. and Thurner,V., Towards a formalization of the unified modeling language. ECOOP'97 procs., Lecture Notes in Computer Science vol.1241, Springer, (1997).

Cockburn, Alistair. Writing Effective Use Cases. Addison-Wesley. 2001

D´ Souza, Desmond and Wills, Alan. Objects, Components and Frameworks with UML.Addison-Wesley. 1998.

Dijkstra, E.W., A Discipline of Programming. Prentice-Hall, 1976.

Evans,A., France,R., Lano,K. and Rumpe, B., Developing the UML as a formal modeling notation, UML'98 Beyond the notation, Muller and Bezivin editors, Lecture Notes in Computer Science 1618, Springer-Verlag, (1998).

Evans,A., France,R., Lano,K. and Rumpe,B., Towards a core metamodelling semantics of UML, Behavioral specifications of businesses and systems, H. Kilov editor, Kluwer Academic Publishers, (1999).

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. November 2002.

Harel David and Kupferman Orna. On Object Systems and Behavioral Inheritance.IEEE Transactions on Software Engineering. Vol.28, No.9. Sept. 2002.

Kim, S. and Carrington,D., Formalizing the UML Class Diagrams using Object-Z, proceedings UML´99 Conference, Lecture Notes in Computer Sciencie

1723, (1999).

Övergaard, G., A formal approach to collaborations in the UML, <<UML>>´99 - The Unified Modeling Language. UML´99 conference, USA,. Lecture Notes in Computer Science 1723, Springer. (1999).

Pons Claudia and Baum Gabriel. Formal foundations of object-oriented modeling notations 3rd International Conference on Formal Engineering Methods, ICFEM 2000, York, UK.IEEE Computer Society Press. September 2000.

Pons, Claudia, Giandini Roxana and Baum Gabriel. Specifying Relationships between models through the software development process, Tenth International Workshop on Software specification and Design (IWSSD), San Diego, California, IEEE Computer Society Press. November 2000.

Pons, Claudia. Generalization relation in UML model elements, Proceedings of the Inheritance Workshop at ECOOP 2002 ISBN 951-39-1252-3.

Pons Claudia and Baum Gabriel. Contracts Soundness for Object Oriented Software Development Process.. OOPSLA'2002 Workshop on Behavioral Semantics.Seattle, Washington, USA. Northeastern University, College of Computer Science, pag. 163-177. November 2002.

UML-OMG. The Unified Modeling Language Specification – Version 1.4, UML Specification, revised by the OMG, http://www.omg.org, September 2001

Wegner, P.and.Zdonik, S, Inheritance as an Incremental Modification Mechanism or What like is an isn't like. in proceedings 3rd European Conference on Object-Oriented Programming (ECOOP'88), Springer, 1988.