OBJECT-ORIENTED MODELING BY VIEWPOINT USING UML

Hair Abdellatif
Faculté des Sciences et Techniques
B.P. 523 Beni-Mellal, Maroc
Tél : +212 23485112 – Télécopie : +212 3485201
– Mél : a.hair@fstbm.ac.ma

Sbihi Boubker
ENSIAS
B.P. 713, Agdal Rabat Maroc
Tél: +212 61556588 – Télécopie : +212 7777230
– Mél : Sbihi7@Hotmail.com

Ettalbi Ahmed
ENSIAS
BP 713, Rabat, Maroc
Tél: +212 377773 1 – Télécopie : +212 7777230
– Mél : Ettalbi@ensias.ma

Abstract : The aims of this work is, firstly to propose a new assimilation of the visibility relationship of VBOOL in the UML (Unified Modeling Language) standard language for modeling and specifying object-oriented systems. Secondly, to complete UML by an oriented viewpoint method to get a complete software engineering process. U_VBOOM method represent an adaptation of VBOOM (View Based Object-Oriented Method) integrating the UML notation. The new U_VBOOM method keeps main VBOOM concepts and integrates to its development process strong points of UML such as use cases and a new stereotype <<interpreted>> that allows with the aggegation relationship to implemente the visibility relationship. This new approach encourages the multi-targets code generation and improve the process of development proposed by the VBOOM method.

Key-words: Method, Analysis/Design, View et Viewpoint, Use case, UML (Unified Modeling Language), Visibility Relationship

1. Introduction

Recently, two aspects have received a lot of attention in object-oriented development: the emergence of the Unified Modeling Language (UML) as a unified notation for object-oriented analysis and design, and integrating viewpoint approaches to software development.

The UML [Jacobson and al., 1999][Omg, 2001] can be seen as the successor of the wave of object-oriented analysis and design methods that appeared in the late 80s and early 90s. It unifies the methods of Booch [Booch, 1994], Rumbaugh (OMT) [Rumbaugh and al., 1995], and Jacobson (OOSE) [Jacobson and al., 1992]. The UML is a standard language for modeling and specifying object-oriented systems. It gives notations for describing a system in various views, but does not define any specific process for software development, beyond some preliminary process description reported, for instance, in [Jacobson and al., 1999].

The introduction of viewpoint approaches to software development provides several improvements in complex system modeling [Bardou, 1998][Carré and al., 1991][Finkelstein and al., 1990][Mili and al., 1999]. In fact, it enables the users to build a unique model accessible by different users with various viewpoints, instead of building several sub-system whose management is too hard to complete. The concept of viewpoints was first introduced by Shilling and Sweeny [Shiling and al., 1989] as a filter of a global interface of the class, but the views are not separable or separately reusable. Harrison and Ossher proposed subject-oriented programming as a way to build integrated "multiple view" applications by composing application fragments, called subjects, which represent compilable and possibly executable functional slices [Harrison and al., 1993][Kaplan and al., 1999]. The approach proposed by S. Marcaillou that interest us more especially consists to define a new language VBOOL (language that extend Eiffel) which integrate the new relation "the visibility" and its derived mechanisms [Marcaillou, 1995]. To implement those concepts in object-oriented methodology, VBOOM (View Based Object Oriented Method) has been defined which extends the BON's method [Coulette and al., 1996][Kriouile, 1999].

The aims of this work is, firstly to propose a new assimilation of the visibility relationship of VBOOL in the UML (Unified Modeling Language) [Jacobson and al., 1999][Omg, 2001][Rumbaugh and al., 1999]. Secondly, to complete UML by an oriented viewpoint method to get a complete software engineering process.

This article is organised as follows: in section 2, we will present briefly the assimilation visibility relationship and its derived concepts in UML. The section 3 deal present principles of the about VBOOM method under the UML standard (named thereafter U_VBOOM). We conclude by a survey of work done and a presentation of its perspectives after a presentation of the relative works.

In this paper, we are going to illustrate our subjects with the Media library Management example. The specifications of this example are more explained in [Lopez and al., 1998]. The **Media library** system must allow its members to consult and to borrow various types of support: books, video and audio disks, audio CD, etc. Only one member of the library can borrow books, reviews, etc. The borrow is limited in time. The potential users of the **Media library** system are: the **librarian** who manages the loans, the **person in charge of adhesions** who will add and withdraw members, the **person in charge of examplaries** who will seize the new examplaries and to withdraw those damaged, and finally the system engineer who ensures the good exploitation of the system for the users. According to the use types, the Media library system will be considered, as a set of ADHERENTS ACCOUNTS, or of EXEMPLARIES, or a means to facilitate the LOANS. Thus, we identify 4 classes of the system: Media_Library, Loans, Exemplaries and Counts_Adherents.

We are going to illustrate the implementation code in C++ language [Stroustrap, 1997]but this remained valid for other object-oriented language.

2. The visibility relationship and its derived concepts in UML

By visibility, we mean that an entity can be seen upon several angles. It's well the fact of a Media Library which can be seen under the loans, exemplaries, adherents accounts, etc. angle. Several solutions of implementation and assimilation has been proposed for example the S. Marcaillou proposition. This proposition consists to assimilate the visibility relationship to selective multiple inheritance in VBOOL object-oriented language [Marcaillou, 1995]. The VBOOL language proposes the flexible class concept (multiview class). This is a class that declares more than two visibility ties with other classes named its "views". A **view** is an abstraction of the model. It constitutes the unity of visibility, it is the result of factorizing user's needs. The instantiation of flexible class consists to specify a particular viewpoint which takes various appearances. In the figure 1, the Media_Library class is a flexible class which owns 3 views (Loans, Exemplaries, Counts_Adherents). But, this assimilation suffers mainly from the non availability and implementation of the VBOOL compiler.

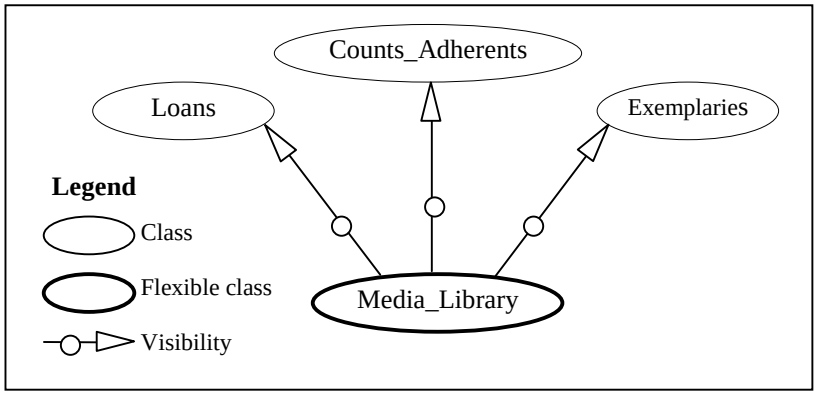

Figure 1: Visibility relationship in VBOOL

We propose a new assimilation of the visibility relationship in UML in order to make object-oriented modeling by viewpoint based on UML and to take advantages of the object-oriented languages like C++, Eiffel, Java, etc. This approach is inspired mainly from the one proposed by M. Nassar [Nassar, 1999] and S. Marcaillou [Marcaillou, 1995] to compile the VBOOL code by an intermediate generation of the Eiffel code [Meyer, 1995].

2.1. Visibility relationship

The approach adopted by M. Nassar [Nassar, 1999] et S. Marcaillou [Marcaillou, 1995] to implement the visibility relationship consists mainly in transforming this relation in selective multiple inheritance. The new approach consists firstly to use the aggregation/delegation relationship instead of the inheritance. Indeed, the delegation mechanism used jointly to the aggregation relationship allows a class to delegate the request treatment to another class. Like the inheritance, the under-class delegue to the sub-class the inherited methods treatment. In the inheritance case, the delegation mechanism is treated automatically by the language while for the aggregation the customer object has access automatically to receiving object through the intermediary of the **self** reference (or by **this** in the C++ case). Secondly, to permit the possible representation evolutivity for object because this last, in some cases, can intervene and evolve in several processes according to the use type played in these processes.

for our example of the *Media library management*, the Media library system can be used and to undergo a different evolution in the management of loans process, in the management of exemplaries and in the management of members one. That is, the Media library system is not discerned by a customer that through an use. It comes back to say that, the Media library system can be used under the differents use types: Management of Loans, Management of Exemplaries and Management of Members. Thus, the Media_Library multiview class delegue the Media library system use as a means of loans to the LOANS class. It's in the same way of the other views, as Exemplaries and Counts_Adherents (Figure 2).

```
# include <Loans.h>                    // include Loans class
# include <Exemplaries.h>              // include Exemplaries class
# include < Counts_Adherents.h>        // include Counts_Adherents class
 ….
Class Media_Library
public:
     loans_view: Loans
     exemplaries_view: Exemplaries
     counts_adherents _view: Counts_Adherents
 ….
End; // end of class
```

Figure 2: Views declaration in a multiview class in C++

The aggregation relationship used in our approach, so much in its realization that in its conceptual taxonomies, is not modified but it is stereotyped by the <<interpreted>> word (Figure 3). This stereotype is added to signal that it is necessary to add to the aggregation class (Media_Library mutiviews class), the primitives who have the export statute restraints to the instances having a specific viewpoint on the aggregation class (Media_Library mutiviews class).

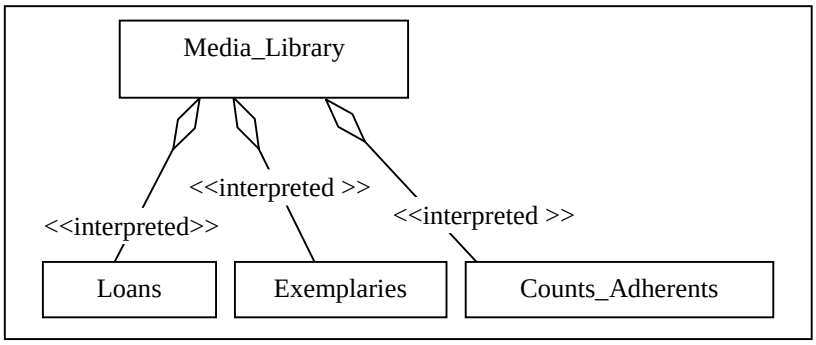

Figure 3: Visibility relationship representation in UML

In the VBOOL language, the multiview class instanciation consist in specifying a particular viewpoint. For example *Media_library_librarian: Media_Library (Loans, Exemplaries, Counts_Adherents)* is an instance of Media_Library having the viewpoint (Loans, Exemplaries, Counts_Adherents). The multiview class instance according to a particular viewpoint in the new proposed approach is transformed to a simple declaration as all other class. But, it is necessary to specify (to activate) the views constitute the viewpoint. For that, we propose to add a mother class, named **View**, of all the classes representing the views of the system. This class has a boolean attribute **view_state (**Figure 4) who return the view state and 2 operations **activate_view** and **desactivate_view** (Figure 4) that permit respectively to activate and to disactivate a view (viewpoint is a combination of the active views).

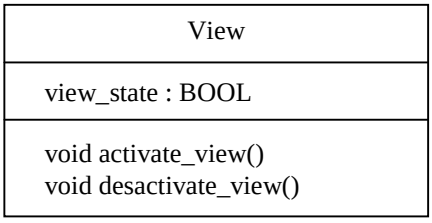

Figure 4: Interface of the View class

The Media_Library multiview class instanciation according to the Media_library_librarian viewpoint amount to the declaration *Media_library_librarian: *Media_Library* and the activation of the three views Loans, Exemplaries and Counts_Adherents by calling the three instructions (the views are initially all disactivated):

```
Media_library_librarian -> loans_view. activate_view();
Media_library_librarian -> Exemplaries_view. activate_view();
Media_library_librarian -> Counts_Adherents_view. activate_view();
```

2.2. Mutual exclusion views

The object to use the viewpoint is to define access rights to the model. If we consider **librarian**'s access for example to our model the views Loans, Exemplaries and Counts_Adherents concern him. Contrary to the the **person in charge of members**, the **librarian** must not have the right to change the adherent's feature "block_count_adherent", so to call this feature of the Counts_Adherents view. To solve this problem, the "mutual exclusion views" concept has been introduced, that means to specify the views that cannot be simultaneously in the same viewpoint. Thus, in the Media library Management example, we can create two views inheriting from Counts_Adherents: the views Mod_Counts_Adherents and Not_Mod_ Counts_Adherents. These two views are in mutual exclusion (Figure 5): that is to say Mod_Counts_Adherents view that will have access the the **person in charge of examplaries** and Not_Mod_Counts_Adherents view that will have access the **librarian.** Of the same way, the **librarian** must not have the right to change the "number_available_exemplary" feature of a Exemplaries view, contrary to the the **person in charge of examplaries** who can add new exemplary or to withdraw the damaged exemplaries by invocation the two features "add_exemplary" or "withdraw_exemplary" of the Exemplaries view. Thus, we can also create two views inheriting from Exemplaries, Mod_Exemplaries view that will have access the the **person in charge of examplaries** and Not_Mod_Exemplaries view that will have access the **librarian** (Figure 5).

In order to not specify simultaneously in the same viewpoint two Mod_Exemplaries (=V1) and Not_Mod_Exemplaries (=V2) views in mutual exclusion, we add a mutual reference in the two views. With the mutual reference of V1 we can reach to the V2 view and can know the **active** or **disactivate** state of V2 and vice versa. To achieve this, we add a mother class, named **Exclusion_View**, of all the classes representing the views of the system. This class has a **exclusion_view** feature of **View** type to maintain the view reference in exclusion.

Mod_ Exemplaries.exclusion_view = Not_Mod_ Exemplaries;
Not_Mod_ Exemplaries. exclusion_view = Mod_ Exemplaries;

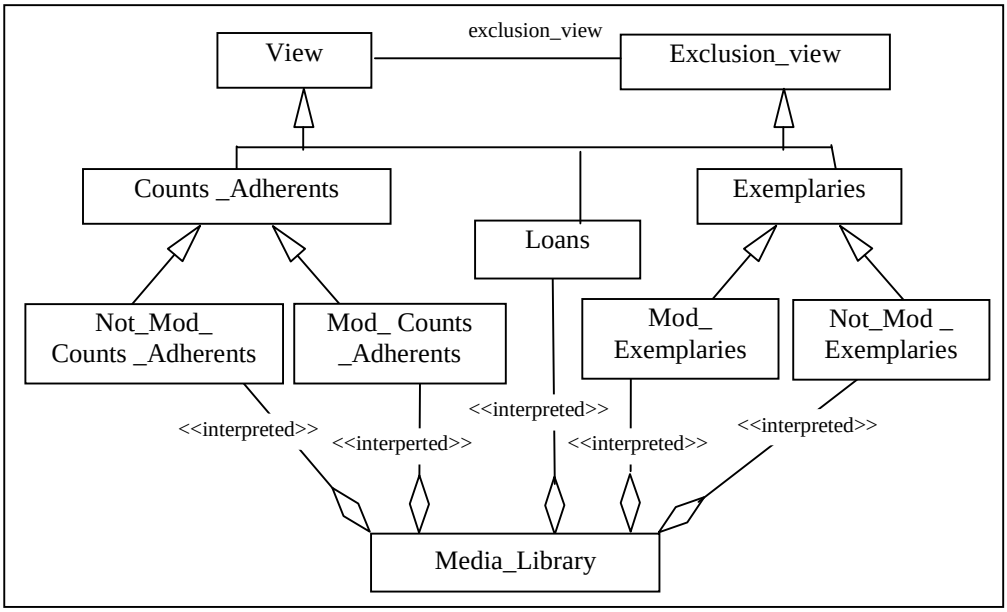

Figure 5: Oriented diagram viewpoint in UML

2.3. Viewpoint evolution Dynamic

The view declaration is static of multiview class, but the viewpoint mechanism is dynamic. The features (attributes and methods) meentioned in the paragraph (II.2) permit the viewpoint evolution applicable to the objects. These features are evidently submitted to strict rules. The use of such primitive allows the user to make evolve dynamically the viewpoints of the objects and give him the possibility therefore to make evolve his access rights on the model. These features will be declared in the View class having the public visibility.

3. U_VBOOM method

VBOOM is a method of analyse/design which integrates the multiview approach in a coherent and deductive method [Coulette and al., 1996][Kriouile, 1999]. VBOOM permits to lead specific partial designs to the different use types of the system. Every use type of the system is associated to a sub-system of the global system. These sub-systems (named as model's view) are melted thereafter as a model global multiview, accessible according to several viewpoints.

Like what has been made for the OMT[Rumbaugh and al., 1995], OOSE[Jacobson and al., 1992], BOOCH[Booch, 1994] methods, the VBOOM method must take in account the standard UML, i.e to integrate the concepts and the notations used in the unified modeling language in the VBOOM method, by using its possibilities of specialization and extension (notably the stereotypes) [Hair, 2000][Hair and al., 2001][Hair and al., 2002]. The U_VBOOM method presents an adaptation of VBOOM to UML. The development model of U_VBOOM is iterative, incremental, and piloted by the use case of UML [Jacobson and al., 1992][Jacobson and al., 1999]. In considering only one iteration, we can divide the analyse/design of a system in three stages.

3.1. Stage1: global analysis

The object of the first stage is to definie model components. It is a stage of global specification of development by U_VBOOM. It consists in providing a precise description of the different needs of the system users.

To express the users needs through the system, U_VBOOM proposes the use cases of UML (Figure 6). The views elaboration of the system is supported by the use cases description technique in actions [Hair and al., 2001][Hair and al., 2002]. This technique consists to describe the use cases as an action sequence wich will make an actor to achieve his goal. An action is an effect produced by an actor acting in way given on the system or by the system itself [Dano and al., 1997]. Every action has a number and a label who are indicated in the "Decomposition in actions" column (Figure 7).

The use cases identified for the actors and the intersection of their actions are going to permit to cut the viewpoints in views. A view corresponds to actions list to a given viewpoint or result of actions intersection of viewpoints group (Figure 8).

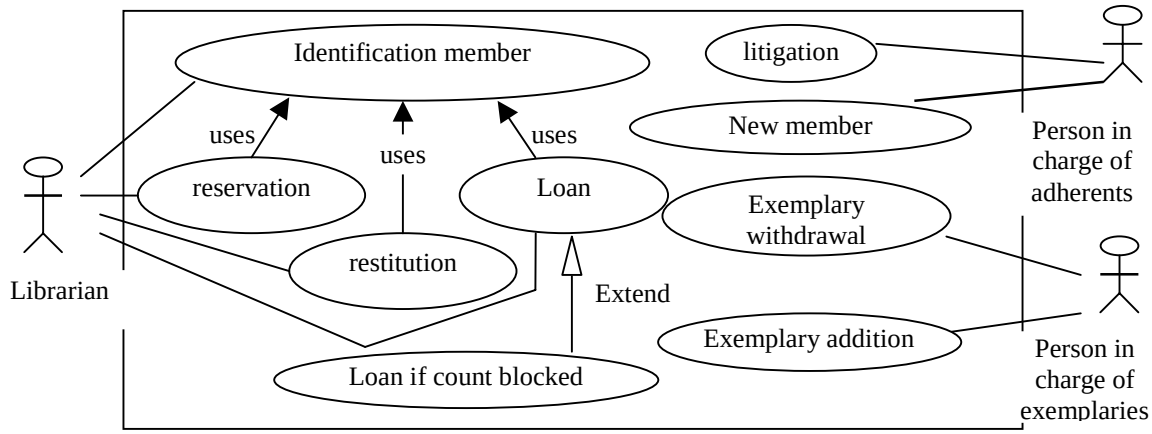

Figure 6: Use case diagram of MEDIA LIBRARY system

Actors	Use case	Decomposition in actions
Librarian	Loan	**a1**: To identify an adherent **a2**: Count adherent (To check right loan for member) **a3**: To seek for an exemplary **a4**: To treat an exemplary (to validate the output of an exemplary) **a5**: To treat adherent (to indicate the loan by adherent)
	Reservation	**a1**: To identify an adherent **a2:** Count adherent (To check right loan for member) **a3**: To seek for an exemplary **a6**: To reserve an exemplary
	Restitution	**a1**: To identify adherent **a3**: To seek for an exemplary **a4**: To treat an exemplary (to validate the input of exemplary) **a5**: To treat adherent (to indicate the return of an exemplary)
	Loan if counts blocked	**a1**: To identify adherent **a2**: Count adherent (To check right loan for member)
	Identification member	**a1**: To identify adherent
Person in charge of adherents	New adherent	**a 2**: Adherent account (to Add adherent)
	Litigation	**a1**: To identify adherent **a2**: Count adherent (Blocked count adherent) **a 7**: To inform adherent
Person in charge of exemplaries	Exemplary addition	**a 3**: To seek for an exemplary **a 8**: To add copy
	Exemplary withdrawal	**a 3**: To seek exemplary **a 9:** To withdraw the damage exemplary

Figure 7: Decomposition of the different use cases in actions

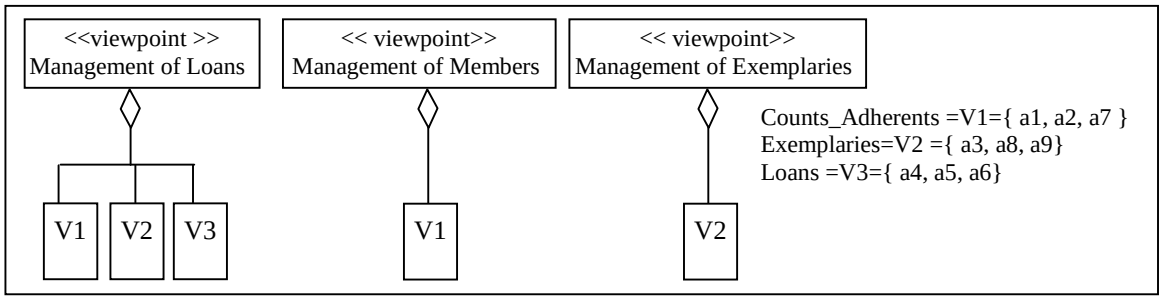

Figure 8: Viewpoint diagrams of the MEDIA LIBRARY system

The global analysis stage continues to identify objects and classes belong to the with problem domain. The classes are discovered, scenarios after scenarios, by means of objects who, in collaborating, achieve use cases. It leads to the development of the class diagrams (Figure 9) and the object diagrams.

Finally, the packages can be identified to organise the modeling elements. The identified packages are gotten by according to the logical criteria use type of an actor. These packages are going to become thereafter the sub-system (named model's view) during the second stage of the U_VBOOM method (Figure 10).

The classes constituting a package represent a part of class diagram of the system. This part is defined in respecting the following rules:

- The only views (classes) of the package are those of its associated viewpoint;
- The classes joined by the customer relationship to one of the package classes;
- The classes jointed by inheritance of each package classes.

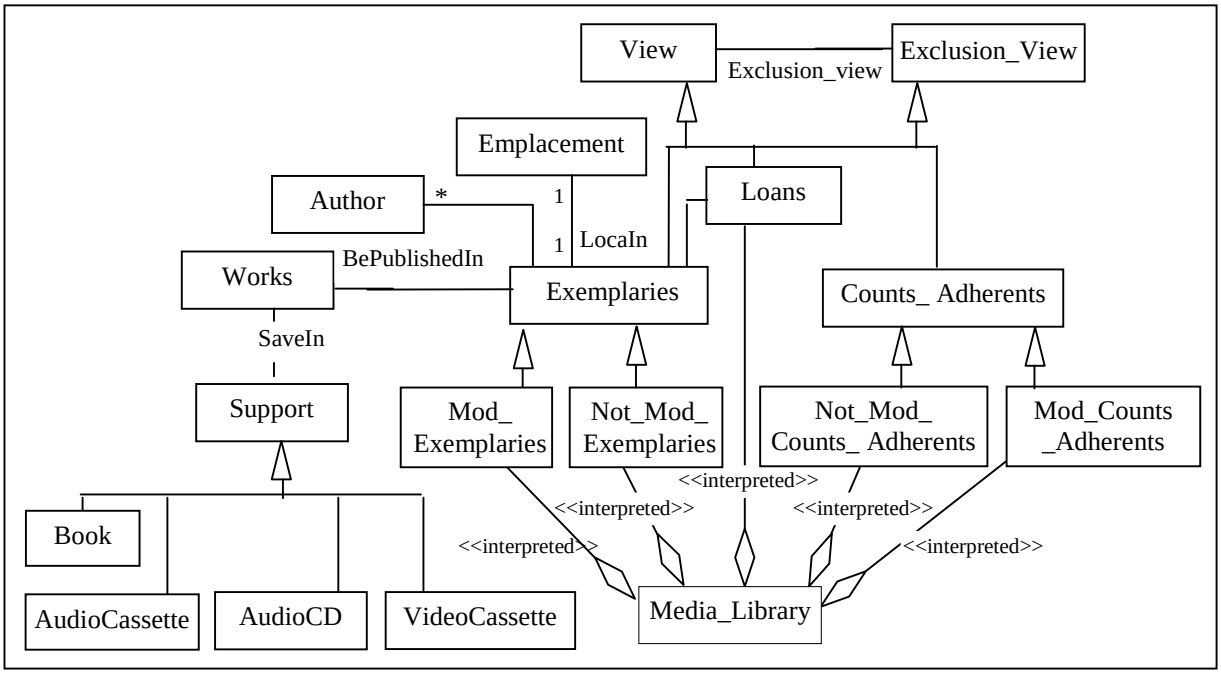

Figure 9: Initial class diagram of the MEDIA LIBRARY system

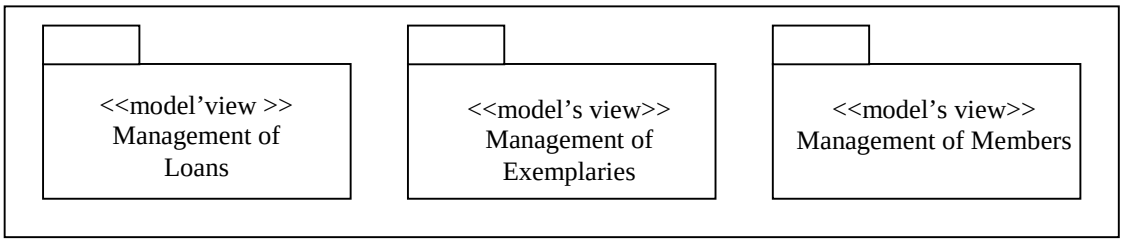

Figure 10: The 3 packages of the MEDIA LIBRARY system

3.2. Stage2: The sub-systems design

The global analysis of the U_VBOOM method is elaborated and it will be enriched in second stage. The design of sub-systems (packages identified in the global analysis stage, named model's view) permits the translation of the analysis model. These sub-systems constitute an essential artifact of the second stage of the U_VBOOM method. Indeed, the cutting out the solution space of the problem in sub-systems permits to land the global system design to sub-systems designs. The design of sub-systems can be made in a disparate and autonomous way and be led in parallel or sequential.

The second stage of U_VBOOM has for object to achieve the partials class diagrams and to define the partials classes interfaces of every sub-systems (the class interface contains the list of its features). The designer must come back toward to realise the use cases. The scenarios, the collaborations between analysis objects and the class diagrams are refined to get the design classes constituting the partial dictionary of every sub-system. The figure 11 represent the class diagram (partial) of the **Management of Loans** sub-system.

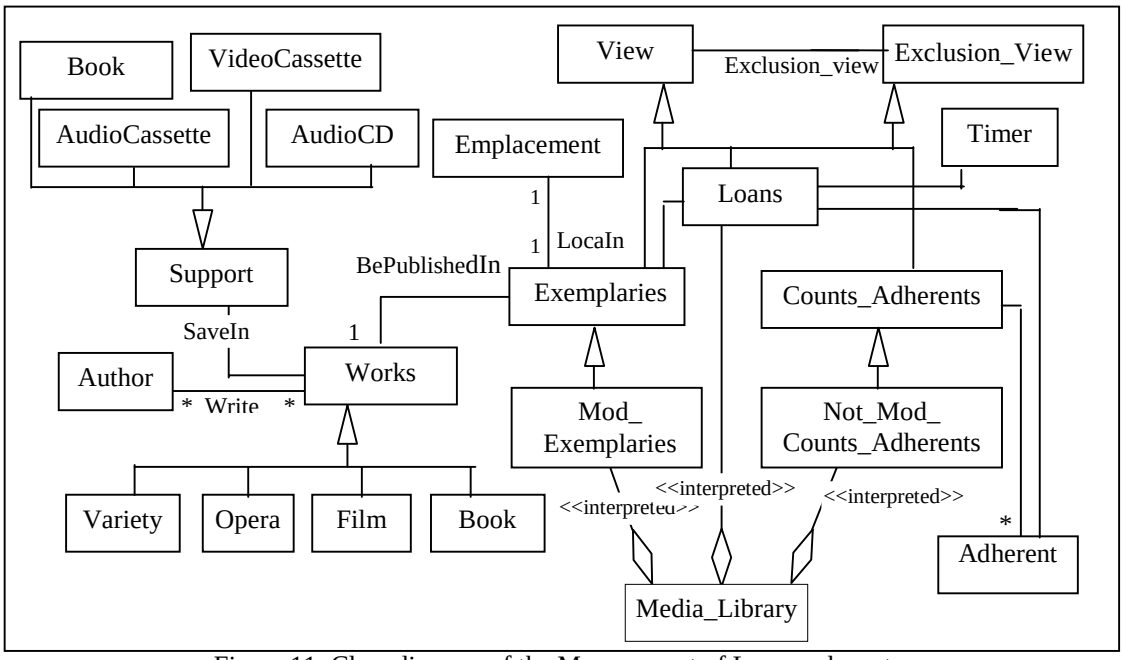

Figure 11: Class diagram of the Management of Loans sub-system

3.3. Stage3: Global model design

The third stage of the U_VBOOM method provides to the process designers to melt the differents partials class diagrams and the differents partials class interfaces of the sub-systems obtainde from the second stage of the method. The melting process proposes to the designers the heuristic to manage the conflicts appeared during of this stage (polysemys, synonymies, homonymies...) (Figure 12). In our example, the global model is relatively close to the **Management of Loans** sub-system because this last is predominant in the system, but this situation is not evidently a generality. The sub-systems obtained are tested and validated in order to be coded in the implementation activity.

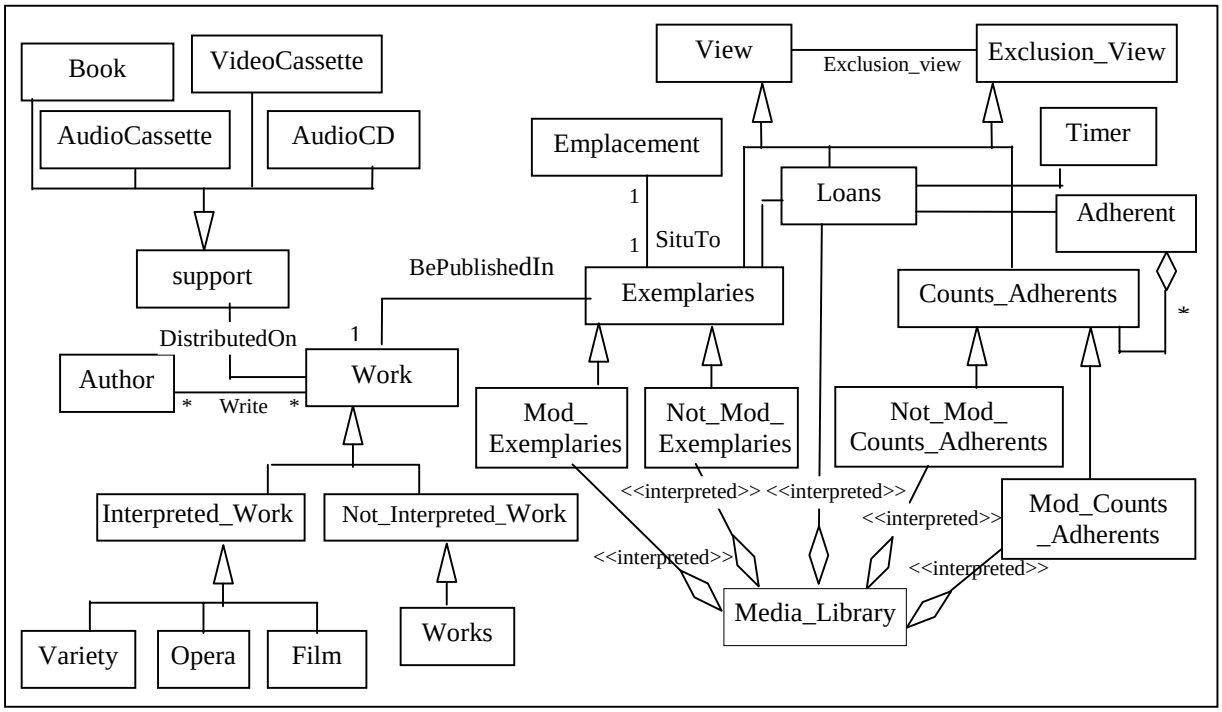

Figure 12: Class diagram of the Media library management

4. Conclusion

The assimilation of relationship visibility of VBOOM in UML to an interpreted aggregation encouraged the integration of UML in VBOOM. The U_VBOOM method that represents the result of the VBOOM adaptation under the UML standard is incremental, iterative and piloted by the use cases.

The decomposition of the system in packages, who became the sub-system, by the logical cutting based on use type permitted to land the global system design to the sub-systems designs. The new approach presented, in this article, of relationship visibility assimilation permitted to solve the multi-targets code generation problem as (C++, Java, Eiffel,etc.) from the UML classes gotten by an oriented viewpoint modeling.

The work describes herein is part of a project which define a methodology of development of components multiview objects. Among the tasks remaining to achieve in this project, we can mention:

- the definition of the composing multi-views notion as regrouping of multiview classes,
- the development of a basis of design pattern supporting the viewpoints approach,
- the realization of an environment support of U_VBOOM.

References

[Bardou, 1998] Bardou, D., (1998) Etude de langages à prototypes, du mécanisme de délégation et de son rapport à la notion de point de vue. Doctorate Thesis in Computer Science, LIRMM, Université de Montpellier2.

[Booch, (1994] Booch, G., (1994) Object-Oriented Analysis and Design with Applications. (Second edition), Benjamin/Cummings, Redwood City.

[Carré and al., 1991] Carré, B., And Geib, J.M., (1991) The Point of View notion for Multiple Inheritance. In Proceedings of the ECOOP/OOPSLA.

[Coulette and al.,1996] Coulette, B., Kriouile, A., and Marcaillou, S., (1996) L'approche par points de vue dans le développement orientée objet de systèmes complexes. L'Objet vol. 2, nr. 4, pp. 13-20.

[Dano, and al.1997] Dano, B., Briand, H., and Barbier, F., (1997) An Approach Based on the Concept of Use Cases to Produce Dynamic Object-Oriented Specifications. In Proceedings of the Third IEEE International Symposium on Requirements Engineering.

[Finkelstein and al.,1993] Finkelstein, A., Gabbay, D., Hunter, A., Kramer, J., and Nuseibeh, B., (1993) Inconsistency Handling in Multi-Perspective Specifications. In Proceedings of the ESEC'93, Garmish-Paternkirchen (D), pp. 84-99.

[Finkelstein and al.,1990] Finkelstein, A., Kramer, J., and Goedicke, M., (1990) Viewpoint Oriented Software Development. Génie Logiciel & Application, Toulouse, pp. 337-351.

[Hair, 2000] Hair, A., (2000) Vers une démarche unifiée basée sur le concept point de vue. Scientific conference on the systems engineering, NîmeTIC2000, Nîme.

[Hair and al., 2001] Hair, A., Kriouile, A., and Coulette, B., (2001) VUML : Une méthode d'analyse et de conception orientée objet, intégrant UML et le concept de point de vue. International Conference on Systems, Software Engineering and their applications, ICSSEA'2001, Paris, France, vol. 3.

[Hair and al., 2002] Hair, A., Kriouile, A., and Coulette, B., (2002) Un processus d'analyse et de conception unifié basé sur le concept de point de vue. Proceeding of the Acte 6th Africain Conference on Research in Computer Science, CARI'02, Yaoundé, Cameroun, pp. 229-237.

[Harrison and al., 1993] Harrison, W., and Ossher, H., (1993) Subject-oriented programming: a critique of pure objects, in Proceedings of OOPSLA'93 ; Washington D.C., pp. 411-428.

[Jacobson and al., 1999] Jacobson, I., Booch, G., and Rumbaugh, J., (1999) The Unified Software Development Process. Addison Wesley, Inc..

[Jacobson and al., 1992] Jacobson, I., Christerson, M., Jonsson, P., and Overgaard, G., (1992) Object-Oriented Software Engineering, A Use Case Driven Approach. Addison-Wesley.

[Kaplan and al., 1995] Kaplan, H.M., Harrison, W., Katz, A., and Kruskal, V., (1995) Subject-oriented composition rules. In Proceedings of OOPSLA'95, Austin, TX, pp. 235-250.

[Kriouile,1995] Kriouile, A., (1995) VBOOM, une méthode d'analyse et de conception par objet fondée sur les points de vue. Thesis in Computer Science, faculté des sciences de Rabat, Maroc.

[Lopez and al., 1998] Lopez, N., Migueis, J., and Pichon, E., (1995) Intégrer UML dans vos projets. Eyrolles Edition.

[Marcaillou, 1995] Marcaillou, S., (1995) Intégration de la notion de points de vue dans la modélisation par objets ; Le langage VBOOL. Thèses de l'université Paul Sabatier, Toulouse, France.

[Meyer, 1995] Meyer, B., (1995) Object success - A managers's guide. Prentice Hall - The Object-Oriented Series.

[Mili and al., 1999] Mili, H., Dargham, J., Mili, A., Cherkaoui, O., and Godin, R., (1999) View programming of OO applications. TOOLS, USA.

[Nassar, 1999] Nassar, M., (1999) Vers une programmation orientée objet par point de vue - Conception et réalisation d'un compilateur pour le langage VBOOL -. Thesis in Computer Science, universite Mohamed V, ENSIAS, Rabat, Maroc

[Omg, 2001] Omg, (2001) Unified Modeling Language (UML), version 1.4. OMG Document formal/2001-09-07, http://www.omg.org/cgi-bin/doc?formal/01-09-67.

[Rumbaugh and al.1995] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., (1995) OMT : Modélisation et conception orientées objet. Prentice Hall.

[Rumbaugh and al.1999] Rumbaugh, J., Jacobson, I., and Booch, G., (1999) The Unified Modeling Language Reference Manual. Addison Wesley.

[Shilling and al.,1989] Shilling, J., and Sweeny, P., (1989). Three Steps to Views. In Proceedings of OOPSLA'89, New Orleans, LA, pp. 353-361.

[Stroustrup, 1997] Stroustrup, B., (1997) The C++ Programming Language. (Third Edition), Addison-Wesley.