AMO - Advanced Modeling and Optimization, Volume 6, Number 1, 2004

The filtering mechanism to the service of the UML

Boubker Sbihi
Ecole des sciences de l'information
BP 6204 Agdal,Rabat,Morroco
+212 66 38 20 45
Bsbihi@Esi.ac.ma

ABSTRACT

The objective of this paper is to present an approach that allows modeling and implementing thereafter an software system with different points of view. The norm defined by the UML language obliges to formulate our solution according to its notation. However, the visibility offered by the UML is insufficient to support the dynamic aspect of points of view. To composite for this problem we defined a mechanism of filtering based on the UML and that permits to filter services offered by a class of a system according to its points of view. Such an addition will permit to complete the UML visibility and implement codes of a complex system by the majority of oriented objects languages of the market as JAVA, C++,...

Keywords

Point of view, UML, Filtering mechanism, Multi-points of view system, Pattern of implementation

1. INTRODUCTION

With the advent of the third millennial, computer system users have became more demanding concerning functionality, richness, diversity of information and conformity to the demands of the various users. It is precisely for this reason that these systems must be evolutionary in order to welcome new users and offer to every one a wide access to data of all nature. Such a modeling cannot be achieved according only to one point of view because of the different needs of every user and his rights of access to relevant information. It is for this reason that several research works of research relative to the point of view's concept in different fields of the data processing have been marked the previous decade. Indeed, the point of view's notion has been studied in correlation with the object's notion in many works interesting various domains of the data-processing research. It is the case of the systems LOOPS and TROPES [26] in the domain of the representation of knowledge [5], of roles models [14] and of the two approaches based on the UML of Clark and Catalysis [6], in the conception of application [8] [11]. It is also the case of systems of data bases object with points of view as O2Views, COCOON and MultiView [26] [1], to only mention some of them. The point of view's notion was also dealt with in the domain of the programming object notably in the programming by subjects [20], the programming by aspects [16], and finally the programming by views [18]. Concerning our works within our team they have already led to the definition of the VBOOL language [17], to the method associated VBOOM and to its extension toward the UML, U_VBOOM [15]. In the same way similar works concerning the generation of code multi-targets for methods VBOOM and U_VBOOM while taking as basis on the filtering mechanism as a basis was the object of some of our research [21] [22] .

The Standard defined in the unified modeling language UML standardized by the OMG (Object Management Group) [19] covers the static and dynamic aspect of a system according

to its different diagrams. The notion of view in UML [27] is a means placed at the disposal of the conceptor to structure a conception: use case, logic, components, deployment. However, the UML alone, doesn't allow to integrate the notion of view within its modeling in diagrams of classes. In this context, it proposes a static visibility that cannot change dynamically with time and that doesn't permit a fine cutting according to the different points of view. It doesn't answer, therefore, to requirements of complex system users. In order to resolve this problem, we have decided to add to the diagram class UML a filtering mechanism [21] [24] which has already been used before to model the dynamic aspect of a non normalized modeling that is the one of the VBOOM's method. In this paper, after an introduction (section1), we will proceed to the definition of the UML visibility for the class diagram (section 2). In the second place we will describe the filtering mechanism based on the UML visibility (section 3); then we will present an example of the use of the filtering mechanism for attributes and methods through a pattern of implementation (section 4, 5, 6, 7), before concluding these works by a conclusion and some perspectives.

2. THE VISIBILITY UML

The unified modeling language UML is the result of the fusion of several methods fusion oriented objects and presents itself as the reference in term of modeling object. It covers the static and dynamic aspect of a system according to its different diagrams. Its goal is to specify, visualize, construct and document the computer systems mainly the complex systems. For that purpose, it defines nine diagrams to represent the different points of view of the modeling. They are presented in the figure below (Figure 1) :

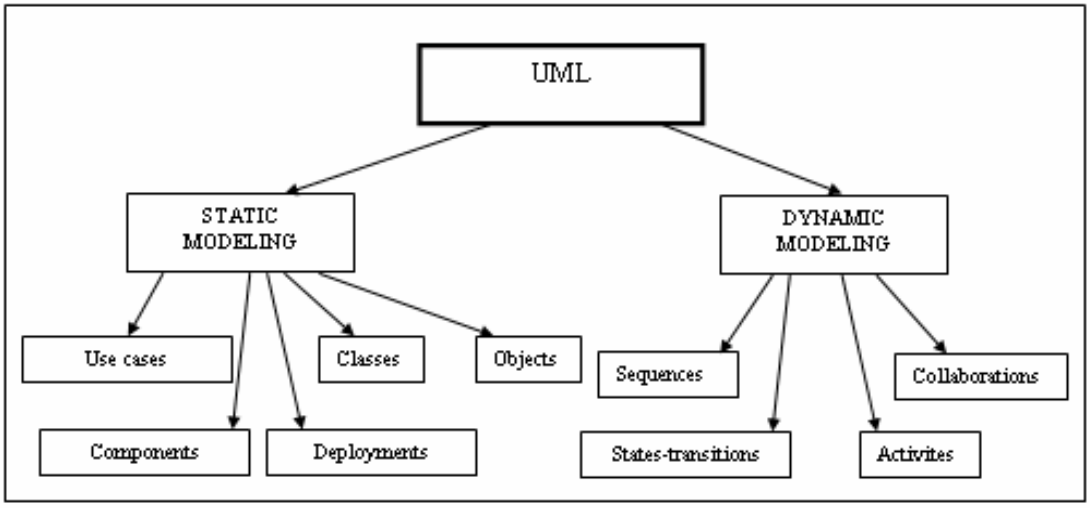

Figure 1. The nine diagrams of the UML

The nine diagrams of UML are subdivided in :

- Static views, that represent physically the system to modelled by means of diagrams of objects, of classes, of use cases, of components and deployments.

- Dynamic views, that show the working of the system by means of diagrams of sequences, collaborations, states transitions and activities.

The diagram of classes is the basic model for the implementation of a software since it can be translated directly in the different programming languages such as Java, C++ or C #. The model object that constitutes the bricks of the software to be achieved can be deducted directly from the model of classes corresponding. Besides, several data processing software exist (Rational Rose,…) which take diagrams of UML classes in charge and generate the code

relating to then. These two diagrams will be concerned by the filtering mechanism that will be applied to the diagram of classes and will be reflected on the relevant diagram of objects. Every class of the classes diagram possesses a set of attributes (state) and a set of methods (behavior). A class in UML is an abstract type characterized by properties (attributes and methods) common to a set of objects and permitting to create objects having these properties (Figure 2) :

Figure 2. Composition of a class in UML

The modeling language UML implements the notion of visibility by distinguishing three levels of visibility on the primitives (Private, Protected, Public). The significance and the UML notation of these visibilities are defined in the following table (Table 1) :

Notation	Significance
- private	Primitive is only visible in the class
+ public	Primitive is visible by all other classes
# protected	Primitive is visible by the class and its under classes

Table 1. Levels of visibility UML and their significance

These three levels are sufficient in the case of a system not requiring to be endowed of points of view. But this is not the case for a system considered according to different points of view, i.e. containing objects that are not seen of the same way by all users of the system. Indeed the UML alone only, cannot even take in consideration the points of views notion for a class. The distinction of three levels of visibility is too restricted to manage points of view. Also, the dynamic change of visibility is impossible during the execution (for example an attribute of UML public visibility will always remain so during the execution).

Moreover, if we associate a level of visibility to an attribute of an object in a system, then all users are obliged to see this object from the same angle, independently of the point of view of each of these users. With regard to attributes, UML defines three properties for their use. These properties are defined in the following table 2 (Table2) :

Property	Significance
Changeable	Attribute in reading/writing
Frozen	Constant
Read Only	Attribute in alone reading

Table 2. Properties of attributes defined in UML

Let's take, for example, an attribute to which we want to give the privilege of reading to certain points of view, the modification for others and to mask its value for the remainder; it cannot be endowed with many properties (Changeable, read-only) at the same time. It result then, the insufficiency of the visibility of the UML.

Therefore, to solve the problem of is the integration of the points of view's notion in the UML, it was necessary to find a means giving the possibility of modeling, dynamically and according to points of view for systems including multi -points of view objects, the visibilities of the primitives. Let's note that our solution that is the one of the filtering mechanism is based entirely on the UML visibility.

3. THE FILTERING MECHANISM

In order to remedy to the problems of the inability of the visibility that UML offers for supporting the dynamic aspect (dynamic change of points of view), we defined a filtering mechanism [21] [22] [23] [24]; this mechanism permits the filtering of the services offered by one class, according to the point of view of the user asking for these services. Indeed, based on the UML visibility, it permits us to define a visibility on a primitive of a class according to the different points of view. It allows us to change the visibility of the primitives dynamically with time and can give to an attribute the possibility of reading to certain points of view, the modification for others and to mask its value for the remainder. In this context, the notion of the point of view consists in making the distinction between the points of view of different users, according to the to the system to be modeled. This distinction appears mainly at the level of the rights of access of each of these users essentially to the different entities or attributes of the system. An user's visibility, in relation with a primitive, determine the level of access of this user on this primitive as for the possibility of his use.

The filtering mechanism targets the primitives like a unit of work. It defines for them a new visibility, allowing thus the management of access rights. In this new context, an attribute could be either inaccessible, or accessible in reading, or accessible in writing according to the points of view. So, it allocates three types of visibility to attributes presented in the table hereafter (Table 3):

Type of Visibility	Value Visibility
Nor reading, nor modification	1
Reading only	2
Reading and modification	3

Table 3. Types and Values of visibility for attributes

The same principle is applied for methods since it forbids or allow the access according to points of view. It also allocates two types of visibility for the methods illustrated in the table below (Table 4):

Type of Visibility	Value Visibility
Not accessible	4
Accessible	5

Table 4. Types and Values of visibility for methods

A UML class diagram of a complex system is not only composed of the primitives divisible between different points of view. It can include public primitives for all the points of view. In order to simplify the filtering mechanism and to optimize its implementation, we will note that a primitive divisible for all type of access (reading and modification for attributes and accesses to methods) will be called public divisible primitive to the different points of view; on the other hand a primitive that is not will be called a multi-points of view primitive.

Therefore, the primitive are organized in the UML in the form of classes. In this context, a class that possesses at least a multi -points of view primitive will be called multi - points class of view ; otherwise, it will be called public divisible class (this one won't be taken in consideration in the filtering mechanism). The goal of the filtering mechanism is to mask the primitives shared by several points of view and to allow the access to them only through predefined filtering methods taking points of view like arguments. In our approach, the brick is the whole object and not only he roles [14] or the parceled out objects [3] or subjects [20] or views [18]. This model offers the advantage to be implemented by any language oriented object as Java, C++, C #,…

4. THE IMPLEMENTATION'S PATTERN OF THE FILTERING MECHANISM

The Patterns of implementation are sometimes called patterns of coding theyn concern implementation advices in a specific programming language. They permit to define how to program in a particular language. They associate to a problem a solution of implementation. We find implementation patterns mainly in C++, in Smalltalk and lately in Java [13]. The complete pattern of implementation would have to, in addition of the solution, clearly explain the problem and the context.

To optimize the UML class diagram of a system with points of view, we will add to this diagram a class called Filtering Class which implements the filtering mechanism. This choice comes from the good reutilisability of code implemented in this class. However, we cannot use this filtering class and implement the methods of filtering in multi-points of view classes. The filtering class possesses a certain number of methods endowed with a public UML visibility UML taking the points of views as parameters. These primitives will be accessible from the class of filtering in multi-points of view classes, through the relation of the UML simple inheritance. To distinguish the filtering class, we will add to him the UML stereotype <<Filtering>>. For the filtering mechanism, every multi-points of view class should be a downward class by inheritance of the filtering class since it is precisely in this one that we are going to implement the filtering mechanism. We illustrate the filtering mechanism with the help of the survey of case of a class STATION that can be seen according to three different view points. Every point of view (Developer, Engineer system, Seller) possesses only one view on the station (V - Developer, V - Engineer, V - Seller) that are respectively relative to him. The diagram UML modeling the Station will be able to be modeled in UML as follows (Figure3):

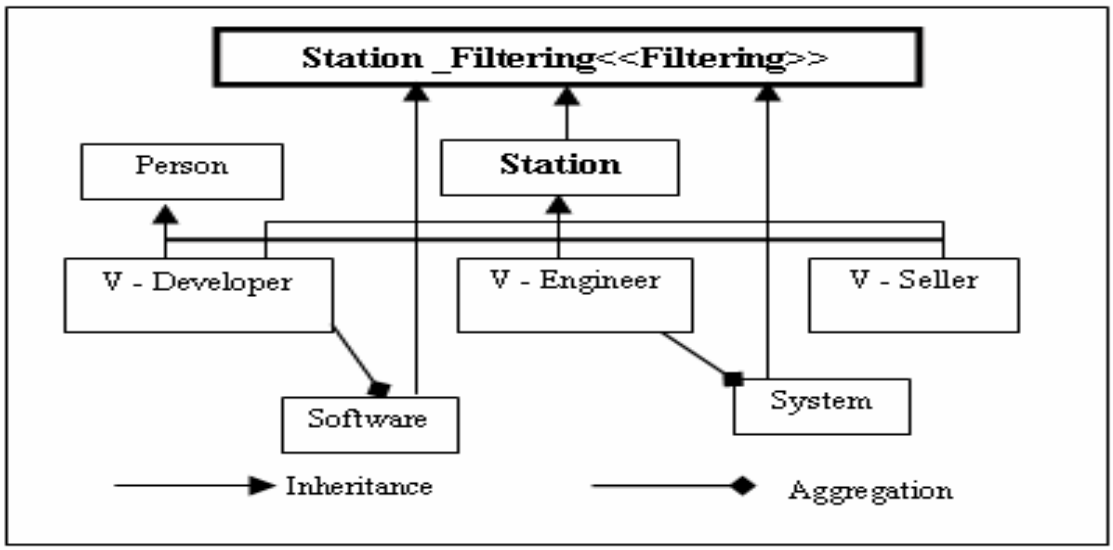

Figure 3 : Diagram of class of the station in UML

In the example of the station, our filtering class is called Station_Filtring..It contains the attributes and the methods of the filtering mechanism. The class Person is a public divisible class that doesn't possess a primitive multi-points of view ; that is why it won't be retained in the filtering mechanism. On the other hand, multi - points classes of view Station, Software and System will be taken in account since they possess the multi-points of view primitive. They must be joined to the Station_Filtring class via the relation of inheritance. To return our approach really usable, we decided to target the languages oriented objects of the market on the one hand, and on the other hand to propose a translation of a model with classes of filterings in a model of implementation multitargets (Java, C++, Eiffel,…). We go, in what follows, to define a pattern of inspired implementation of Gamma patterns [12]. It has for goal

of implementing the notion of points of view in a diagram class UML that is, in our case, the one of the station. It is applied mainly to classes of languages oriented objects.

5. DICTIONARY OF PRIMITIVE MULTI - POINTS OF VIEW

In order to implement the filtering mechanism, we will define a new point of view that is the one of the administrator of points of view. He has the possibility to add, to modify, to suppress the other points of view and their views in a dynamic manner. He must be provided however of a password to do that. The administrator of points of view possesses are 3 for every multi-points of view attribute and 5 for every multi-points of view method. The multi-points of view's primitives of the example of the station can be gathered in a table named dictionary of the multi-points of view primitives. This dictionary illustrates the points of view and their rights of access on the global multi- points of view diagram that appears in attributes and multi-points of view methods. For our case, we can have the following dictionary (Table 5):

N° AND NAME OF THE CLASS		N° AND NAME OF THE PRIMITIVE		N°OF THE POINTS OF VIEW (*)			
				0	1	2	3
1	Station	1	Mark	3	2	2	2
		2	Speed	3	2	2	2
		3	Start()	5	5	5	5
2	V-Developer	1	Name	3	3	1	1
		2	Outputdev()	5	5	4	4
3	V-Engineer	1	Name	3	1	3	1
		2	Num_ip	2	1	2	1
		3	Proc_install ()	5	4	5	4
		4	Outputeng()	5	4	5	4
4	V-Seller	1	Price _HT	2	1	1	2
		2	Rate_TVA	2	1	1	2
		3	Name_ supplier	2	1	1	2
		4	Maintien (date)	5	4	4	5
5	System	1	Name	2	1	2	1
		2	Version	2	1	2	1
		3	DisplayVersion ()	5	4	5	4
6	Software	1	Name	2	2	1	1
		2	Version	2	2	1	1
		3	DisplayName ()	5	5	4	4

Table 5. Extract of the dictionary of primitives multi - points of view

(*): 0: Administrator, 1: Developer, 2: Engineer, 3: Seller.

6.FILTERING AND CALL TO METHODS

In order to implement the values of the dictionary of the multi-points of view primitive, we will add to the filtering class the following attributes and methods :

6.1 THE ATTRIBUTES OF POINTS OF VIEW'S ADMINISTRATION

There are two attributes of points of view administration ; a table of character String will model the different points of view and a matrix with three dimensions of whole type (int)

will model the visibility of these points of view. It will have a parameters, the number of the point of view, the number of the class and the number of the attribute or the method of multi-points of view class. Their UML visibility is declared private to forbid the direct access without the slant of these methods.

NAME	UML VISIBILITY	TYPE	SIGNIFICANCE	
Point of view	Private	String []	Table of points of view	
Visibility	Private	Int [] [] []	The 3 dimensions of the matrix	Number point of view
				Number of the class
				Number of the Primitive

Table 6. The declaration of administration's attributes of points of view in Java

We can mention, a example, for the point of view fields and the visibility:

Point of view [0] = "Administrator" and Point of view [1] = Developer ".

Visibility [0][1][2]=3 // the visibility of the point of view Administrator on the class Station on multi-points of view attribute Speed is equal to 3.

The administrator of points of view can easily add and can dynamically extract points of view and their visibilities while modifying values of administration attributes.

6.2 THE METHOD OF ACCESS TO THE VISIBILITY AND METHODS OF ACCESS TO THE PRIMITIVES MULTI-POINTS OF VIEW

To reach the visibility of a multi-points of view primitive, according to a specific view point, we will use the Getvisibility method that possesses the parameters of whole type (int).

NAME METHOD	UML VISIBILITY	DESCRIPTION	TYPE RETURN	PARAMETERS
Getvisibility	Public	To give the value of the visibility	Int : (1,2,3) Attributes Int : (4,5) Methods	N° Point of view
				N° Class
				N° Primitive

Table 7. Declaration of the access method to the visibility by points of view in Java

To reach the visibility of a primitive multi-points of view, according to a specific view point, we will use the following UML public visibility methods :

NAME METHOD	DESCRIPTION METHOD	TYPE RETURN	TYPE AND NAME PARAMETERS	
AccessNameAttribute Access NameMethod	Return the name of the multi - points of view primitive	Class	Int	N° PV
AccessTypeAttribute	Return the type of the multi -points of view attribute		Int	N°Class
AccessTypeReturn Method	Return the type of return of the multi-points of view method			
AccessReadAttribute	Return the been worth of multi-points of view attribute		Int	N°Primitive
			String	Password
AccessWriteAttribute	Change tea been worth of the attribute and return the been worth 0 so mistakes, 1 otherwise,	Int	Int	N° PV
			Int	N°Class
			Int	N°Primitive
			String	Newvalue
			String	Password
AccessTypeParametre Method	Return the type of parameters	Class []	Int	N° PV
			Int	N°Class
			Int	N°Primitive
			String	Password
InvokeMethod	Reach the method with these parameters and return the been worth of the new object so not of mistake otherwise it is the hopeless	Object	Int	N° PV
			Int	N°Class
			Int	N°Primitive
			Object []	TypesParametres
			Object	Parameters
			String	Password

Table 8. Declaration of access methods to the primitives multi-points of views in Java

6.3 THE METHODS OF EVOLUTION OF POINTS OF VIEW

In order to manage the points of view and their visibilities, we will define six methods of management; three among them (AddPV, ModifyPV, SuppressPV to) are relative to points of view, and the others (AddVisibility, ModifyVisibility, SuppressVisibility) to their visibilities (Table 9). These methods can be only used, by the administrator of points of view that possesses the adequate. privilege to make this task. They possess a UML public visibility to allow the administrator of points of view to reach them while specifying his identity.

THE NAME	UML VISIBILITY	DESCRIPTION	TYPE RETURN	TYPE AND NAME PARAMETER	
AddPV	Public	To add a point of view	Int : 1 no mistake Int : 0 otherwise	String	New PV
				String	Login Administrator
				String	Password Administrator
ModifyPV	Public	To modify a point of view	Int : 1 no mistake Int : 0 otherwise	String	New PV
				Int	Old PV
				String	Login Administrator
				String	Password Administrator
SuppressPV	Public	To suppress a point of view	Int : 1 no mistake Int : 0 otherwise	Int	Point of view
				String	Login Administrator
				String	Password Administrator
Add Visibility	Public	To add a visibility for a point of view on a primitive of a class	Int : 1 no mistake Int : 0 otherwise	String	Login Administrator
				String	Password Administrator
				Int	N° Point of view
				Int	N° of the class
				Int	N° of the primitive
				Int	New value
Modify Visibility	Public	To modify a visibility of a point of view on a primitive of a class	Int : 1 no mistake Int : 0 otherwise	String	Login Administrator
				String	Pass Administrator
				Int	N° Point of view
				Int	N° of the class
				Int	N° of the primitive
				Int	New value
Suppress Visibility	Public	To suppress a point of view on a primitive of a class	Int : 1 no mistake Int : 0 otherwise	String	Login Administrator
				String	Password Administrator
				Int	N° Point of view
				Int	N° of the class
				Int	N° of the primitive
				Int	New value

Table 9. List of methods of management of points of view and visibilities

The administrator will use the AddrPV method to add a new point of view in the filtering Class while giving as parameters: the new point of view, the name of the administrator point of view and his password. In the same way, he can use the other methods pertaining to points of view. He will use the AddVisibility method to add a new visibility on a class of the diagram primitive; the administrator must provide the following parameters: the administrator login, the administrator password, the number of the point of view concerned, the number of the class, the number of the primitive, the new value to be added. In the same way, he can use the other methods pertaining to visibilities of points of view.

7.CALL OF PRIMITIVES MULTI-POINTS OF VIEW

The instanciation of a new object belowing to a multi-points of view class of the class diagram or no, won't change the syntax. However, it will change at the time of the affectation or the use of an attribute or to the call of a multi-points of view method.

The access will be made according to the three methods of the accessible filtering mechanism from the class of filtering while using the relation of inheritance. In what follows continuation, we give some examples of the use of these methods:

To reach, in reading, a multi-points of view attribute according to a specific point of view as for example, the multi-points of view attribute « Speed » in the class Station by the developer, we use the AccessReadAttribute method that returns the value of the multi-points of view primitive and that possesses as parameters: N° Point of View, N°Class, N°Primitive and password. The call of this method in java is made as follows :

Station S= new Station();

float Value_Speed=S.AccessReadAttribute(1, 1, 2, "Password_ developer") ;

To reach in writing a multi-points of view attribute according to a point of specific view as for example, the « Num_ip » multi - points of view attribute in the class V-Engineer by the engineer who possesses the right to change it in an address of value : 172.1.5.7, we use the AccessWriteAttribute method ; this method changes the value of the attribute and return the value 0 in case of mistake and 1 if there is no mistake. It possesses as parameters: N° Point of View, N°Class, N°Primitive, New value and password. The call of this method is made as follows :

V-Engineer VE=new V-Engineer();

Int Value =VI.AccessWriteAttribute(2, 3, 2, "172.1.5.7", "Password_engineer") ;

To call a method according to a specific point of view, for example the call of the multi-points of view method of Maintien(Date) in the classe V-Seller by the seller, we will call the InvokeMethod method ; this method reaches a multi-points of view method while giving as parameters: N° Point of View, N°Class, N°Primitive, TypesParametres, Parameters and password. It returns the value of the new object if is not mistake and the null object in case of mistake. The call of this method is made as follows :

V- Seller VS=new V-Seller();

Object o= VS.InvokeMethod (3, 4, 4, Date, D Date, "Password_seller").

In the same way, we can use the remainder of the methods illustrated in table n°9.

The model UML as result by points of view guarantees the coherence of data, the suppression of certain redundancies, the management of access rights, the centralization of the knowledge and the possibility to evolve dynamically the points of view. The filtering mechanism defines a filtering on attributes and methods while taking the UML visibility like a basis. It forbids the direct access to the primitive and gives back the possible access to them by the slant of methods predefined that have points of view as arguments (methods of filtering). It targets the multi-points of view primitive as a unit to which it defines a new visibility allowing the management of access rights.

CONCLUSION

The interest approach presented in this article is to integrate concepts of point of view in a diagram of UML classes in order to automate the generation of the code source of complex applications with the majority of the languages oriented objects while remaining in the standard. It essentially rests on the notion of interface that is a kind of filter on all

functionalities offered by one given class. It also serves to make points of view of a complex system evolve dynamically. The filtering mechanism gives a net progress in the modeling of complex technical systems encouraging the conherence, the reliability and the reutilisability of the produced models. We have also defined a pattern of implementation which describes the generation of code object relative multitargets to a multi-points of view component.

The work describes here, is part of a wide project. The objective of which is to define a methodology for the development of multi-points objects of view components. Among the tasks that are still to be performed :

- The definition of the multi-points of view component notion, as applicable regrouping of classes possibly multi-points of view.

- The development of a basis of pattern of conception supporting the approach by points of view

- The standardization of the U_VBOOM method.

REFERENCES

[1] Abiteboul S.and Bonner.A, 1991, Objects and Views. *Proceedings of ACM SIGMOD, p. 238-247.*

[2] ASJ. AspectJ-Oriented Programming (AOP) for Java. *http ://www.aspectj.org*.

[3] Bardou.D. 1998. *Survey of languages to prototypes, of the delegation mechanism, and of its report to the notion of point of view*. Thesis of doctorate, computer specialty, University Montpellier 2.

[4] Bouraqadi.N and Ledoux.T. 2001, The point on the programming by aspects. *Technique and Computer Sciences*, vol 20, page 505 à 528, Hermés.

[5] Carré.B and Geib.J.1991,The Point of View Notion for Multiple Inheritance, *Proceeding Of ECOOP/OOPSLA.*

[6] Clarke.S.2002. Extending standard UML with model composition semantics. *Science of Computer Programming, Elsevier Science*, 2002.

[7] Coad.P 1992, Object-Oriented Patterns.*Communications of the ACM*, vol 35, N°9, p. 152-159.

[8] Coulette.B and al.1994. *The* approach by points of view in the development oriented object of the complex systems. *Reviewed the object*, vol 2, n°4, p. 13-20.

[9] Couturier.V and Séguran.M. 2003. Patterns and Components to Capitalize and Reuse a Cooperative Information System Architecture. *In Proceeding of the 5th International Conference on Enterprise Information Systems ICEIS'03, Angers, 23-26 p. 225-231.*

[10] Debauwer.L and al. 2000. Contextualization of OODB Schemas in CROME. *DEXA 2000, 11th International Conference*, London.

[11] Finkelstein.A and al.1993. Inconsistency Handling in Multi-Perspective Specifications. *Actes conférence ESEC'93*, Garmish- Patemkirchen (D), pp. 84-99.

[12] Gamma.B and al..1995. *Design patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley.

[13] Grand.M.1999. *Patterns in Java*. Volume 2, Wiley & Sons. Tropes. 1995. *Tropes 1.0 reference manual*, INRIA Rhônes-Alpes IMAG LIFIA, Grenoble, France.

[14] Gottlob..G and al.1996. Extending Object-Oriented Systems with Roles. *ACM Transactions on Information Systems (TOIS)*, page 268-296.

[15] Hair.A, Sbihi.B and A.Ettalbi.A. 2003. Object-oriented Modelling by viewpoint using UML, *Advanced Modeling and Optimization*, pp.107-115, http://www.ici.ro/camo/journal/v5n2.htm

[16] Kiczales.G and al.1997.*Aspect-Oriented Programming. In European Conference on Object-Oriented Programming (ECOOP)*, Finland, Springer-Verlag LNCS 1241.

[17] Marcaillou.S.1995.*Integration of the notion of points of view in the modeling by objects - The Language VBOOL*, Thesis of the university Paul Sabatier of Toulouse.

[18] Mili.H and al.2000. *Views*: A Framework for Feature-Based Development and Distribution of OO Applications. *Proceedings, Thirty-Third Hawaii International Conference on System Science*, Honolulu, HI.

[19] OMG. http://www.omg.org

[20] Ossher.H *and al.1996. Specifying subject oriented composition. I*in Theory and Practice of Object Systems (TAPOS), 2(3), 1996. Special issue on Subjectivity in Object-Oriented Systems.

[21] Sbihi.B and al.2003. Toward a generation of code multi-targets for the VBOOM method : Approach by filtring, *The 2003 International Conference on Software Engineering Research and Practice (SERP'03)*, Las vegas, USA.

[22] Sbihi.B and al.2003. Implementation in UML of the points of view's notion in a Distance Education System. *Act 4th International Conference on Information Technology Based Higher Education and Training*, Marrakech, Morocco.

[23] Sbihi.B. 2004. L'utilisation du mécanisme de filtrage pour implémenter un système d'enseignement à distance. *Workshop e-learning vers un campus virtuel marocain*, 8-10 Janvier, Agadir, Morocco 2004.

[24] Sbihi.B and al.2004. The integration of the points of view notion in UML. *Act The IADIS Applied Computing 2004 conference*, Lisbon, Portugal 2004.

[25] Rundensteiner.A.1994. A Classification Algorithm for Supporting Object-Oriented Views. *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, Gaithersburg, Maryland, USA, ACM Press., pages 18-25.

[26] Tropes 1995. Tropes 1.0 reference manual . *INRIA Rhônes-Alpes IMAG-LIFIA*, Grenoble, France.

[27] UML 1.4. Unified Modeling Language, version 1.4, 2003, *http://www.omg.org/docs/ formal/03-03-01.pdf*