VIEWS, SUBJECTS, ROLES AND ASPECTS : A COMPARISON ALONG SOFTWARE LIFECYCLE

Bouchra El Asri , Mahmoud Nassar[*,**], Abdelaziz Kriouile
**Laboratoire de Génie Informatique*
ENSIAS, BP 713 Agdal, Rabat, Maroc

Bernard Coulette
*** Laboratoire GRIMM – IRIT*
Université de Toulouse le Mirail, Département de Mathématiques- Informatique
5, allées A. Machado 31058 Toulouse cédex, France

Keywords: Complex system, Perspective, View, Viewpoint, Subject, Role, Aspect, Software lifecycle criteria.

Abstract: To face the increasing complexity of software systems and to meet new needs in flexibility, adaptability and maintainability, classical object-oriented technology is not powerful enough. As pointed out by many authors, one must take into account the multiplicity of actors' viewpoints in complex systems development. Views, subjects, roles and aspects are viewpoint-oriented concepts that permit a flexible adaptation of modelling and use of systems. This article aims to provide software developers with a comparison between view, subject, role and aspect approaches in respect to their principles and impacts on systems development as well as on systems use. After a brief presentation of these approaches, we discuss their similarities and differences by means of criteria positioning them along the software lifecycle.

1 INTRODUCTION

Nowadays, companies must face complexity and rapid changes of software systems. Profiling, flexibility, reusability, adaptability, interoperability, maintainability and integrity are the main keywords of overcoming challenges. A number of researches in software modelling have attempted to meet those needs. We believe that the conjunction "information-actor" is the strategic gateway to reach this hope. Indeed, actor-centred technologies allow developers to concentrate on those parts of the process and domain models that will be important for their job.

Researches in software modelling and development have spawned various concepts related to *view* and *viewpoint* concepts. The *view* concept was first introduced by Shilling and Sweeny (Shilling et al., 1989) as a filter on global interface of a class. This concept has been then largely investigated in the field of databases (Abiteboul et al. 1991, Debrauwer 1998), Software Engineering (Finkelstein et al. 1990), Requirement Engineering (Charrel, 2002) and Object-oriented development (Carré et al. 1991, Coulette et al. 1996, Vanwormhoudt 1999, Coulondre et al. 1999, Motsching-Pitrik 2000, Nassar et al. 2003). In UML (OMG, 2001), the view notion is a way of structuring system modelling according to development progress: use cases, logical, components and deployment views.

The viewpoint notion was introduced under closely related terms such as role (Anderson et al., 1992), subject (Harrison et al., 1993), and more recently aspect (Kiczales et al., 1997), etc.

Harrison and Ossher (Harrison et al., 1993) proposed *subject-oriented programming* as a way to build integrated multi-perspective applications.

Role (Anderson et al., 1992) and Role modelling (Kristensen et al. 1996, Riehle et al. 1998, Gottlob et al. 1996) were proposed to express and to abstract objects interaction and change.

Introduced by Kiczales et al. (Kiczales et al., 1997), AOP (*Aspect Oriented Programming*) aims to model non-functional concerns into aspects.

Our team has been working on the elaboration of a view-based object-oriented methodology since 1993. We have defined a view-based extension of

El Asri B., Nassar M., Kriouile A. and Coulette B. (2004).
VIEWS, SUBJECTS, ROLES AND ASPECTS : A COMPARISON ALONG SOFTWARE LIFECYCLE.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 139-146
DOI: 10.5220/0002650201390146

Eiffel called VBOOL (Marcaillou et al., 1994) and a view-based analysis and design method called VBOOM (Kriouile, 1995). We are working now on VUML (Nassar et al., 2003), a view-based extension of UML that provides the concept of MultiViews component whose goal is to store and deliver information according to user viewpoints.

Objectives of Mili et al.'s approach (Mili et al., 2000) are quite similar to VBOOM's ones. For those authors, an object can be described as a basic object and a variable set of views representing functional facets that can be added or removed dynamically to reflect the changing roles of the object during its lifetime.

This paper results from a recent study of the state of art in the field of viewpoint-oriented approaches. This study took place in the context of a French-Morocco network in Software Engineering.

In contrast to two similar works – a comparison with a focus on reuse (Bendelloul et al., 2000), and a position paper from Bardou (Bardou, 1998) – our goal is to provide developers with an assessment of such viewpoint-oriented approaches all along the software lifecycle. Hence, we discuss similarities and differences among those approaches according to two sets of criteria: development and run-time criteria.

This paper is structured as follows : the 2nd section describes the view, subject, role and aspect approaches; the 3rd section presents similarities and differences among them according to given criteria. We conclude in section 4.

2 VIEWPOINT-ORIENTED APPROACHES

In this section, we describe a set of viewpoint approaches. Our list is not exhaustive but it covers the most representative approaches within viewpoint concept.

2.1 Subject-Oriented approach

Subjectivity as a way of object-oriented programming was introduced by Harisson and Ossher (Harrison et al., 1993). It allows to express a set of specifications and behaviours shared by several actors. Subject is defined as "a collection of state and behaviour specifications reflecting a particular *gestalt*, a perception of the world at large, such as seen by a particular application or tool" (Harisson et al., 1993).

A subject is not a class but a class inheritance hierarchy where each class defines a structure of its instances' properties and behaviours. A class may appear in different subjects. A subject is an abstraction that can be instantiated in several domains to obtain executing instances. Each instance includes the actual data manipulated by a particular subject.

An object can activate several subjects simultaneously. The essential characteristic of subject-oriented programming is that different subjects can be separately defined and operate upon shared objects. All active subjects share object identities. The universe of system is the composition of all active subjects done in respect to subject composition rules. It reflects the composition of several application slices representing separate functional domains. With that composition it is possible to extend subjects and to introduce new subject activation without disrupting others. The composition (Ossher et al., 1995) consists of (i) the union of the interfaces emanating from the composed subjects, (ii) and the composition of the implementations of the methods that are defined in more than one subject.

2.2 Role approach

The main objective of the role concept is to hold change of object behaviour during its lifetime. In other words, the role concept permits the original classification of an object to change in time (Pernici 1990, Kristensen 1996). A role is a temporary view on an object. Role is expressed by extrinsic features that may change during lifetime. (Kristensen et al., 1996) define a role as a "set of properties which are important for an object to be able to behave in a certain way expected by a set of other objects". For Riehle et al. (Riehle et al., 1998) a "role type describes the view one object holds of another object". An object may play several roles at the same time. Therefore, roles may express the participation of an object to accomplish an activity (Kristensen 1996, Andersen et al. 1992, Andersen 1997).

To explicitly define what the role concept is for, Kristensen introduces a new notion, *roleification* as an abstraction way to express :

Dynamic classification: an object can be dynamically reclassified changing its role during lifetime.

Non-generalization: a role is not a specialization of its corresponding object but it exists together with it in a dependence way.

Identity: a subject (object with currents roles) holds a unique identity even if changing roles during its lifetime.

Extension only: a role can only add extrinsic features to an object but cannot remove or change any intrinsic ones.

Multiplicity: an object may hold several roles at the same time.

Andersen also introduced role models. Role models are much like traditional activity diagrams. In OORAM (Reenskaug, 1995), role models include entities and behaviours that are relevant to a particular collaboration. (Riehle et al. 1998) define a role model as a description of a set of collaborating objects which focus on a single purpose. Thus, role models provide a kind of separation of concerns.

Role models can be composed. A role model composition is a role model in which the individual role models interact according to role type constraints.

2.3 View programming approach

To support the decentralized development of object-oriented applications, Mili et al. propose the *view programming* approach, in which a viewpoint is defined as a generic template that abstracts functional behaviour independently from any domain (Mili et al., 2000). A view is an instance of a viewpoint for a particular domain. Typically, an object may support a set of functionalities or views. Each object of an application is seen as a set of core functionalities (core object) that are available to all the users of the object, and a set of slices (views) that are specific to particular users. Views may be added or removed during run-time. The set of views "attached" to an object determines its behaviours and the messages to which it can respond, and the way it responds to them. Mili el al. propose a set of mechanisms to manage attachment/detachment and activation/deactivation of views.

The model proposed is a set of objects (core and view objects). In such model, sharing features is implemented by delegation mechanism while sharing behaviours is managed by dispatching mechanism. The invocation of methods supported by several views is processed by composition rules. The response to a message depends on the views currently attached to its core instance. The dispatching mechanism is inspired from the composition rules proposed by Harrison and Ossher which consist in combining the different implementations (Harisson et al. 1993, Ossher et al. 1995).

Furthermore, Mili et al. propose a run-time composition of views. Composing them on-demand during run-time allows objects to change their behaviours in non-predictable ways.

2.4 View-based Unified Modelling Language (VUML)

VUML (Nassar 2003, Nassar et al. 2003) is a view-based analysis/design method. VUML provides a modelling language (UML extension) and a process that allows a view-based modelling from analysis to implementation. The main new feature of VUML is the concept of MultiViews component whose goal is to encapsulate and deliver information according to the user profile. A MultiViews component is a unit of abstraction and encapsulation composed of a *default view* (base) - common part of the entity accessible by every actor (end user or not) - and a set of *views* (extending this common part) representing actors' needs and rights. Each view corresponds to one actor. View activation (linkage to the current user's viewpoint) is done at execution time. Views management (add, suppress, lock, unlock) is done dynamically through an implicit administrative view (Nassar et al., 2002). Views are related to the base through a *view-extension* relation which is a dependency relation. It is not an inheritance relation because one cannot create an instance of a view independently from an instance of the default view.

A MultiViews component may have sub-components that become automatically multiviews. Views of the parent component become views of the child one. It is yet possible to define new views on the sub-component or to redefine a parent view.

Furthermore, VUML supports the dynamic change of viewpoints and offers mechanisms to manage views dependencies and maintain the internal coherence of a MultiViews component. To achieve that goal, VUML offers a *view-dependency* relation to make explicit declaration of dependencies between views, and OCL (Object Constraint Language) expressions that are attached to dependency relations.

2.5 Aspect-Oriented Software Development (AOSD) approach

Aspect-oriented approach (Kiczales et al., 1997) aims to modularise non-functional aspects during software developement. The core assumption of aspect programming is that there are concerns which cannot be cleanly encapsulated in traditional object structure and therefore the resulting code is tangled Such concerns are called aspects. Typical examples of aspect are performance, security, logging, synchronization, optimisation, persistence, etc. Usually, such concern interweaves with many objects. This orthogonal intersection is called crosscutting concern. Aspect-oriented programming

(AOP) is a technology for separation of crosscutting concerns into aspects. Kiczales et al. distinguish between component and aspect because, contrary to a component, an aspect cannot be cleanly encapsulated into a generalized procedure (Kiczales et al., 1997).

Aspect weaver is the underlying infrastructure which process the component and aspect languages composing them properly to produce the desired total system operation.

To support such environment, there are different available techniques. The most popular are AspectJ and HyperJ. AspectJ is an aspect language which offer mechanism to modularise and compose crosscutting concerns. It supports aspects as entities that contain join points used to change class definition. HyperJ developed by IBM is an offspring of subject programming providing weaving tools. Hence it is considered as aspect oriented programming.

3 COMPARATIVE STUDY ALONG SOFTWARE LIFECYCLE

As seen in the previous section, several approaches deal with the viewpoint notion, quite often under closely related terms. Main advantages of viewpoints are: (i) reducing complexity of the development by focusing on special portions of systems according to developer skills; (ii) improving accuracy, simplicity, and access right management in respect to user profiles.

Comparing different viewpoint approaches will then concern the two states of software product: development and use. In this section we study differences and similarities between viewpoint approaches according to a well known set of development and use criteria.

3.1 Development criteria

For development process, we have identified the following criteria : use of viewpoints along the development process, reusability, code understandability, and testability.

Use of viewpoints along the development process: we believe that supporting viewpoints throughout the development process is extremely important. Indeed, it permits safe tractability, high reusability, efficient comprehensibility, non-tangled deliverables and so on. So, we study the integration of the viewpoint notion from different approaches in different stages of development process:

Analysis stage: Among viewpoint approaches, studied in this paper, VUML is the only one (in our knowledge) that proposes a process to explicitly identify actors and their needs in the analysis stage. VUML introduces the notion of user viewpoint at the very beginning of the analysis stage since a viewpoint is associated to each actor (end user or not) of the system. This enables developers to identify the right requirements and avoid non-needed features.

Design stage: Supporting viewpoints at design stage allows designers to modularise the system modelling. Furthermore, using CASE tools to visualize and check viewpoint-based models can widely increase the efficiency of software development.

In subject-oriented design (Clarke et al., 1999), a design subject describes only pieces of software concerning a given perspective. The integration of subject models builds the complete design.

Role modelling within design (Riehle, 1998) allows to produce frameworks with well-defined boundaries and defines how to use it with the help of free role types of free role models.

Designing in Mili et al.'s approach (Mili et al., 2000) uses classical aggregation association to model application objects. An application object consists of a core object to which views can be added or removed during run-time. At design time all views are aggregation-based linked components.

The first VUML process phase (analysis) results in a set of UML models (one model per actor). At the design level, those models are melted together into a global VUML model made of *MultiViews components*. VUML addresses the model consistency preservation issue by offering mechanisms to specify dependencies among views during that design phase.

For AOSD, (Suzuki et al., 1999) propose a number of new stereotypes to express aspect class, weave operation and woven classes.

Implementation: A number of approaches have been proposed to support viewpoint coding. We distinguish those which introduce new programming concepts and keywords as Role components (VanHilst et al., 1996), Subject-Oriented Programming (SOP) (Harrison et al., 1993), Aspect-Oriented Programming (AOP) (Kiczales et al., 2001), View programming (Mili et al., 2000), and those which use classical object-oriented programming concepts such as VUML (Nassar 2003, Nassar et al. 2003). We also make distinction between compile-time and execution-time. SOP and AOP integrate concerns automatically at compilation time while view programming (Mili et al. 2000) allows dynamic change of views. Finally, we consider approaches that support code generation.

As far as we know, VUML distinguishes from the others in that it provides a generic implementation pattern to generate object-oriented code (Java, C++, Eiffel) from a VUML model (Nassar et al., 2002).

Reusability: It is an essential quality criterion that concerns all stages of the development lifecycle.

The role modelling does not encapsulate viewpoint into an entity of abstraction. But the distinction done between intrinsic properties and those called extrinsic ones permits the reusability of intrinsic ones.

Mili's and subject approaches take into account the actor's perspective from the design phase. A perception can be applied on several domains enhancing therefore the reusability of models and code.

VUML introduces actors and thus viewpoints during the first step of development. Each actor is therefore elicited, its needs are analysed, conceived and implemented. Hence, deliverables (analysis documents, models, code) regarding an actor can be easily reused.

AOP allows the encapsulation of non-functional requirements of a system as aspects that may be woven and reused by a system soliciting them.

Code understandability: The process of software evolution and updating addresses repairing of defaults, enhancing functionalities, and adding object interactions. Experience asserts that half of evolution cost consists of code understanding and comprehension.

Modelling with subjects, roles, views or viewpoints permit to describe systems into comprehensible models and programs.

AOP whose goal is to avoid tangled code is probably the best approach to write clean and understandable code.

However, reverse engineering remains a complex task for all the approaches.

Testability: Profiling allows different customers with different skills to focus upon their domains. This focus allows deep black box testing since tester are bounded to the customer's domain of interest. On the other hand, the separation of different actors allows a good code understanding that facilitate the white box tests. We can thus deduct that viewpoint-oriented approaches (view, role, subject) favour a better testability.

AOP approach allows focusing upon the functional tests in a first time letting the quality and non-functional concerns for a second time.

3.2 Run-time criteria

For software use we have identified the following criteria: profiling and access right management, dynamism, multiplicity, identity and integrity, and maintainability.

Profiling and access right management: We can define profiling as a means of information accuracy and access right management. Each actor holds a profile that specifies its available data, functionalities and visualisation needs and so on. The viewpoint notion as defined by different approaches is an obvious way for profiling. Software are specified, designed and coded according to different perspectives. AOP is an exception since it is a means of separation of non-functional concerns.

Dynamism: Activation/deactivation mechanisms proposed by Nassar et al. (Nassar et al., 2003) and Mili et al. (Mili et al., 2000) allow dynamic evolution of profiles. In SOP, the choice of the system universe (set of active subjects) is done at compile-time. No run-time evolution is allowed.

Multiplicity: In a distributed system, different actors with different profiles may access different views of an object simultaneously. Mili et al. (Mili et al., 2002) propose DOC (Distributed Object Configurator) as a tool to choose the set of visible views and to manage the methods dispatching as done by subject composition. Nassar et al. (Nassar et al., 2003) manages this multiplicity by a management view tool. Role modelling allows an object to play several roles at the same time. The invocation of the right subject method is guided by the context notion.

Identity and integrity: All profiles, perspectives and contexts share the same set of object identities (differently classified). In VUML, SOP and role modelling, different views of an object are extension of the core object and do not hold any identity.

In Mili et al's approach, both the core object and view have an identity. Views are components that are added or removed to/from the application object according to needs.

In AOP, aspects are not components. An aspect does not hold any identity. However, an aspect crosscut several objects (several identities).

Maintainability: Most software engineering is software evolution or maintenance. This phase represents a considerable amount of lifecycle costs.

In our study we take in account both evolution and corrective maintenance. Evolution or scalability is insured by the application of the viewpoint notion. Indeed, addition of new needs related to a given actor, or addition of a new actor does not disturb the system since viewpoints are separated. The viewpoint is a means of reducing scalability cost. The corrective maintenance can reverberate back on all the levels of development lifecycle. So the earlier actors are taken into account within the development process, the less is the maintenance cost. AOP allows writing clean code that is easily maintainable.

3.3 Recapitulative tables

We summarise our comparative study in the following two tables. Table 1 proposes an assessment of development process criteria for the approaches described above. In table 2, the focus is put on execution-time criteria. For both tables, cells are filled with stars that represent the quality degree of each criterion.

Table 1: Recapitulative table synthesising development process criteria

	Use of viewpoints along the development process			Reusability			Code under-standability	Testability
	Analysis	*Design*	*Implementation*	*Analysis*	*Design*	*Implementation*		
SOP	-	***	**	-	***	**	**	**
Roles	-	**	*	-	**	*	**	**
VP	-	***	**	-	***	**	**	**
VUML	***	***	**	**	**	**	**	**
AOP	-	*	***	-	**	***	***	**

Table 2: Recapitulative table synthesising run-time criteria

	Profiling and access right	Dynamism	Multiplicity	Identity and Integrity	Maintai-nability
SOP	***	-	***	***	**
Roles	**	-	***	***	*
VP	***	***	***	**	**
VUML	***	***	***	***	**
AOP	-	*	-	-	**

Legend: *** : Strongly; ** : Moderately; * : Weakly; - : Not supported or not described in the literature
Acronyms: SOP: Subject-Oriented Programming; VP: View Programming; VUML: View-based UML; AOP: Aspect-Oriented Programming

4 CONCLUSION

Undoubtedly, concepts of viewpoint or related notions help decentralised development, enhance reusability, improve information accuracy and consistency, facilitate code understanding and reduce test time in software production. These concepts can be used however in many ways. View, role subject and aspect are the main viewpoint-oriented approaches. In fact, each approach offers more or less advantages all along the software production lifecycle.

In this paper, we presented these approaches and compared them thanks to lifecycle criteria. Our goal is to provide software developers with an assessment of those approaches as objective as possible.

Separation of functional concerns allows different skilled developers to deeply focus on their job in a decentralized way. Subject, role and view based approaches integrate this kind of separation during the design stage and provide mechanisms to co-ordinate and compose separated concerns. The originality of VUML is that it provides a process to

support this functional separation of concerns in a consistent way from analysis to implementation.

Separation of non-functional concerns allows developers to firstly discard quality requirements and orthogonal aspects to better focus on their job's core.

Both functional and non-functional separations make deliverable products (documents, models and code) cleaner, therefore facilitate code understanding and enhance maintainability.

Static composition of subjects, which are provided by role, subject and view programming approaches, and the weave of aspects, enhances reusability.

Dynamic view composition provided by Mili's approach allows dynamic change of system universe.

Dynamic activation and deactivation of view proposed and managed by Mili et al. and Nassar et al. permit run-time change of system behaviour.

The role, subject and view approaches allow profiling, which enables access right management and information relevance, and multiplicity that permits distributed and coherent access to system.

Aspect-oriented programming does not take explicitly into account users but favour a type of modularised development that meets some of the quality criteria mentioned above. In this respect, this approach is complementary from view-oriented ones.

REFERENCES

Abiteboul S., Bonner A., 1991. Objects and Views. *Proc. of ACM SIGMOD,* pp. 238-24.

Aldawud O., Elrad T., Bader A., 2001. A UML Profile for Aspect Oriented Modeling. *OOPSLA 2001 workshop on aspect Oriented Programming*.

Andersen E. P., Reenskaug, 1992. System Design by Composing Structures of Interacting Objects. *Proc. of the 6th European Conference on Object-Oriented Programming (ECOOP'92)*, LNCS, Vol. 615. pp. 133-152, Utrecht, The Netherlands. Springer-Verlag.

Andersen E. P., 1997. Conceptual Modeling of Objects. A Role Modeling Approach. *PhD thesis*, Department of Informatics, University of Oslo.

Bardou D., 1998. Roles, Subjects and Aspects: How do they relate? *Position paper at the Aspect Oriented Programming Workshop. 12th European Conference on Object-Oriented Programming (ECOOP '98),* LNCS, vol. 1543, Springer.

Bendelloul S., Mili H., Dargham J., Mcheick H., 2000. A comparison of view programming, aspect-oriented programming, subject-oriented programming from a reuse perspective. *Proc. of 13th ICSSEA*, Volume 4, Paris, France.

Carré B., Geib J.M. , 1991. The Point of View Notion for Multiple Inheritance. *Proc. of ECOOP/OOPSLA.*

Charrel P.J., 2002. The Viewpoint Paradigm: a semiotic based Approach for the Intelligibility of a Cooperative Designing Process. *Australian Journal of Information Systems,* Vol. 10, n° 1. pp. 3-19.

Clarke S., Harrison W., Ossher H., Tarr P., 1999. Separating Concerns throughout the Development Lifecycle. *Proc. Of ECOOP'99 Workshop on Aspect-Oriented Programming*, Lisbon, Portugal.

Coulette B, Kriouile A., Marcaillou S., 1996. L'approche par points de vue dans le développement orienté objet des systèmes complexes. *Revue l'Objet vol. 2, n°4,* pp. 13-20.

Coulondre S., Libourel T., 1999. Viewpoints Handling in an Object Model with Criterium-Based Classes. *In DEXA'99, Florence*. LNCS n°1677, pp. 573-582.

Debrauwer L., 1998. Des vues aux contextes pour la structuration fonctionnelle de bases de données à objets en CROME. *Thèse de Doctorat,* LIFL, Lille.

Finkelstein A., Kramer J., Goedicke M., 1990. *Viewpoint Oriented Software Development*. *Proc. of Software Engineering and Applications Conference*, pp. 337-351, Toulouse.

Gottlob G., Schrefl M., Roeck B., 1996. Extending object-oriented systems with roles. *In ACM Transactions on Information Systems,* vol. 14 n. 3, pp. 268-296.

Harrison W., Ossher H., 1993. Subject-oriented programming : a critique of pure objects. *Proc. of OOPSLA'93*, Washington D.C., pp. 411-428.

Kiczales G., Lampng J., Mendhekar A., Maeda C., Lopes C. V., 1997. Aspect-Oriented Programming. *Proc. of the European Conference on Object-Oriented Programming (ECOOP)*. Finland. Springer-Verlag LNCS 1241.

Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., Griswold W.G., 2001. An Overview of AspectJ. *Proc. of ECOOP'*, Springer Verang LNCS2072.

Kriouile A, 1995. VBOOM, une méthode orientée objet d'analyse et de conception par points de vue. *Thèse d'Etat*. Université Mohammed V de Rabat.

Kristensen B. B., 1996. Object-oriented modeling with roles. *In John Murphy and Brian Stone, editors, Proceedings of the 2nd International Conference on Object-Oriented Information Systems*, pages 57–71. Springer-Verlag.

Kristensen B. B., and Osterbye K., 1996. Roles : Conceptual Abstraction Theory & Practical Language Issues. *Theory and Practice of Object Systems (TAPOS)*, 143-160, Special Issue on Subjectivity in Object-Oriented Systems.

Marcaillou S., Coulette B., Kriouile A., 1994. Visibility : A new relationship for complex system modelling. *In TOOLS USA'94*. TOOLS13, Prentice Hall.

Mili H., Dargham J., Mili A., 2000. Views: A Framework for Feature-Based Development and Distribution of OO Applications. *Proc. of 33rd Hawaii Int. Conference on System Sciences*. Honolulu, HI, January 4-9.

Mili H., Mcheick H., and Sadou S., 2002. CorbaViews – Distributing Objects that Support Several Functional Aspects, *in Journal of Object Technology*, vol. 1, no. 3, Special issue : TOOLS USA 2002 proceedings, pp. 207-229.

Motschnig-Pitrik R., 2000. The Viewpoint Abstraction in Object-Oriented Modeling and the UML. *International Conference on Conceptual Modeling (ER 2000)*. Salt Lake City, Utah, USA.

Nassar M., Coulette B., Kriouile A., 2002. Vers un langage de modélisation unifié supportant les vues. *Rapport IRIT 02-29-R*. Octobre 2002, Toulouse.

Nassar M., 2003., VUML : a Viewpoint oriented UML Extension. *Proc. of the 18th IEEE International Conference on Automated Software Engineering (ASE'2003)*, Doctoral symposium, Montreal, Canada.

Nassar M., Coulette B., Crégut X., Ebersold S.., Kriouile A., 2003. Towards a View based Unified Modeling Language. *Proc. of 5th International Conference on Enterprise Information Systems (ICEIS'2003)*, Angers, France.

OMG, 2001. Unified Modeling Language, version 1.4; http://www.omg.org/cgi-bin/doc?formal/01-09-67

Ossher H., M. Kaplan M., Harrison W., Katz A. and Kruskal V., 1995. Subject-oriented composition rules. *In Proc. of OOPSLA'95*, Austin, TX, pp. 235-250.

Pernici B., 1990. Objects with roles. *Proc. of the ACM--IEEE Conference on Office Information Systems*, Cambridge, MA, 1990.

Reenskaug T., 1995. Working with Objects : The OORAM Software Engineering Method. Englewood Cliffs: Prentice Hall.

Riehle D. and Gross T., 1998. Role Model Based Framework Design and Integration. *Proc. of the Conference on Object-Oriented Programming Systems, Language, and Application (OOPSLA '98)*. ACM press, pp. 117-133.

Shilling J., Sweeny P., 1989. Three Steps to Views, *Proc. of OOPSLA'89*, New Orleans, LA, pp. 353-361.Suzuki J., Yamamoto Y., 1999. Extending UML with aspects : Aspect support in the design phase. *In Proc. of the third ECOOP Aspect-Oriented Programming Workshop*.

VanHilst M. and Notkin D., 1996. Using Role Components to Implement Collaboration-Based Designs. *in OOPSLA'96*, San-Jose, CA, 6-10 Oct., pp. 359-369.

Vanwormhoudt G., 1999. CROME : un cadre de programmation par objets structurés en contextes. *Thèse de Doctorat en Informatique,* LIFL, Université des Sciences et Technologies de Lille.