International Journal of Computer Applications (0975 – 8887)
Volume 135 – No.2, February 2016

A New Software Component Approach

Abdellatif Hair
LMACS
P.B. 523, Faculty of Sciences and Technology,
Beni Mellal, Morocco

ABSTRACT
The "component" paradigm appeared in response to boundaries of the object approach. It introduced a new method for designing software applications. This method is based on the assembly of prefabricated software entities called components. The Technology of connector is thus proposed to facilitate assembly. We present in this paper a model of multi-views software components for systems based components. A multi-views software component allows each user to manipulate a system with a vision own to its needs. The assembly of the components in this model is implemented by a new type of connector named visibility connector.

Keywords
Software component, Assembly, Viewpoint, Visibility connector.

1. INTRODUCTION
Building applications has many advantages especially in terms of code reuse, quality of development, modularity, etc. Component Based Software Engineering (CBSE) addresses to improve the development of systems as assembly of software components, components as reusable entities, maintenance and upgrading of systems by customizing and replacing such components [22][23]. Recent years have seen the emergence of different models to components such as EJBs (Enterprise Java Beans) [5] or CCM (CORBA Component Model) [11]. These models facilitate the construction of applications by assembling software components defined by their required and provided interfaces. Despite the hype about these models, the academic community offers many new models to facilitate the specification, construction, deployment or reconfiguration of applications based on software components.

Furthermore, everyone agrees to recognize the user interest in the development of complex systems. The viewpoint concept is an appropriate means to implement this concern [2][8]. This concept has been addressed in the area of programming including programming by topics [15], aspects [14] and by object [20][16]. It was also studied in the case of systems such as TROPES LOOPS [29] in the field of knowledge representation, role models [13] and also in O2Views, Multi-view and COCOON systems [7] for databases.

The introduction of the viewpoint concept in object oriented modeling of complex systems can elaborate a unique model that is shareable and accessible by several viewpoints. The advantage of this new approach appears at the consistency of data, deletion of some redundancy, enhancement of the multi-model approach and the definition of access rights.

Several approaches have been proposed in the object viewpoint modeling, among the most successful works, is the work of [28][1] which led to the definition of the VUML profile (View based Unified Modeling Language). VUML proposes formalism and methodology to support view-based modeling from analysis to coding. VUML enables to model a software system according to each actor's viewpoint. First, actors of the system are identified as in UML. Each actor is associated with a unique viewpoint. Then, for each viewpoint, we describe, in an iterative way, use cases and scenarios as well as related classes. The result is a set of class diagrams (called also viewpoint models) in the UML formalism. Finally, a VUML model is produced by composing the partial models.

Recently J. Krumeich, et al. proposes also a method for viewpoint-based modeling using recommender system in a multiple-user environment [17][18]. The viewpoint based modeling method aims at solving such problems by introducing and using stakeholder-specific viewpoints on collaboratively created models. Our work in this area allowed to standardize the analysis/design method VBOOM (View Based Object-Oriented Method) developed by A. Kriouile, [9] in UML notation [25]. The VBOOM method integrates the viewpoint approach in the object-oriented development. The new standardized method is called U_VBOOM [3][4].

By integrating the viewpoint concept in the CBSE, we will propose a model to facilitate the development of applications based on software components. This model that we present allows the construction of a software component called "Multi-view Software Component" **CLM** by assembly of a software component (Base Component: **BC)** and software components (Views Components: **VC**s). The specificity of a CLM is the possibility to have interfaces whose accessibility and behavior change dynamically depending on the current user (Stakeholder). Its mean we can enable/disable Views Components at runtime. We introduce the visibility relationship in software component models. The reification of this relationship is achieved by the new type of connector: visibility connector.

This work proposes a model of CLM for multi-view systems based on components. The assembly of the components in this model is made by the visibility relationship to be able to design components called multi-view software components. We present in the second section the definition and properties of relationship visibility. In the third, we describe the CLM model and the concepts Base Component (BC), View Component (VC) and visibility connector. Then we present in the fourth section, the process of transformation and implementation of CLM in CCM [11]. The enhanced version of language IDL3 (Interface Definition Language) with the statement of CLM is then proposed and is called IDL3-VIEW [24]. Finally we end with a conclusion.

In this paper, we are going to illustrate our subjects with the Media library Management example. 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 four classes of the system: Media_Library, Loans, Exemplaries and Counts_Adherents.

2. VISIBILITY RELATIONSHIP: DEFINITION AND PROPERTIES

2.1 Definition

By visibility, we mean that an entity can be seen upon several angles. It's the Media Library which can be seen under the loans, exemplaries and adherent's accounts. Several solutions of implementation and assimilation have 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 [27]. The VBOOL language proposes the flexible class concept (multi-view 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).

The visibility relationship is a relationship between an entity (base entity) with its views (views entities) by a set of visibility links.

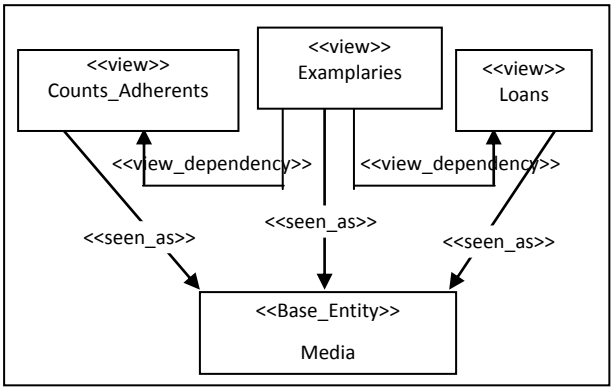

Fig 1: Graph of visibility

2.2 Properties

Some the most essential properties of this relationship are:

Transitivity: Visibility relationship is transitive when the same types of visibility relationship are involved in the premises.

Anti-symmetry: A base entity is seen as an entity view, but the reverse is not possible. For example: the base entity "Media" is seen as an entity view "Loans" but the entity "Loans" cannot be seen as an entity view "Media";

Dependence: This is an important property that allows the changes propagation between the dependent view entities. The propagation direction can be that of the source view entity to the destination view entity (the loan of a book made by a member changes the number of copies, and both views entities "Loans" and " Exemplaries" are dependent);

Mutual exclusion: This property consists of achieving the access right on a multi-view model. Activating a view entity V1 having a mutual exclusion with another view entity active V2 can occur only if the entity view V2 is disabled and vice versa.

3. MULTIVIEWS SOFTWARE COMPONENTS MODEL

We dedicate this section to present our proposal for the multi-views software components model. We begin firstly with the software component basics presentation. And secondly by introducing the visibility relationship for the software components by offering new concepts such as the base component, view component and visibility connector. And we end this section by presenting the multi-views software components model.

3.1 Basics of a software component

This software component is a composition entity specifying, by contract, its interfaces (provided and required) and explicit dependencies contexts. A software component can be deployed independently and can be an assembly element for the software applications design [24].

The software component architecture specifies its inputs and outputs to facilitate the description of his behavior -offered services- regardless of the programming languages used (Figure 2).

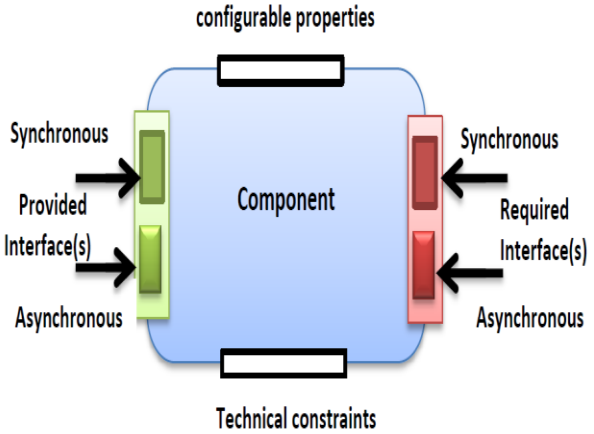

Fig 2: Software component Architecture

As this figure above, a software component has mainly the following three elements:

- The provided interfaces (outputs) and the required interfaces (inputs), in synchronous or asynchronous mode are the means employed to cooperate. These means may be operations (functions promised to customers) or properties;
- Configurable properties: these are attributes generally; they can adapt and customize the component in specific contexts executions;
- Technical constraints (QoS: Quality of Service) can be: the security, persistence, transactions, etc.

3.2 Visibility relationship in software components

To introduce the visibility relationship in software components, we must start giving precisely the definition of a multi-view software component and all visibility relationship properties for software components.

A Multi-view Software Component (CLM) is a software component that can be considered and apprehended under several "angles", named the views components. A multi-view software component consists of a base component (software component) linked with visibility links to views components.

From this definition we can establish the rules to follow in order to introduce the CLM concept in Component Based Software Engineering.

Rule 1: a CLM is primarily a software component that publishes provided and required interfaces.

Rule 2: a CLM consists of a base component and the views components. A base component is linked to its views components by visibility links.

Rule 3: the visibility links should reflect the visibility relationship properties such as dependency, mutual exclusion, etc.

Rule 4: a view component can be applied only on a single base component.

Rule 5: the view components are directly accessible as long as they are not part of a CLM. From the moment they become views components of a CLM, they will be inaccessible except through their CLM.

Rule 6: a CLM should provide a possibility to change dynamically its behavior and its accessibility by the type of use (customer need).

3.3 Presentation of CLM model

In "component" approach, a software application is built by a collection of software components interconnected using connectors [10][26].

In our model, the visibility link is translated into a connector called visibility connector. This visibility connector is a software component. It has its interfaces and interposed between the base component and the view component to ensure the visibility relationship semantics (Figure 3). Therefore, the visibility connector must ensure the visibility relationship properties and particularly two properties: dependency and mutual exclusion.

Dependency: To ensure the dependency between two views components, an integer variable will be stored in the corresponding visibility connector. This variable must be tested for all operations of modifications concerning changes related between two views components.

Mutual exclusion: To ensure the access right between two views components in mutual exclusion, an integer variable will be stored in the corresponding visibility connectors. This variable must be tested for any activation operations for a view component in mutual exclusion.

The CLM use requires the definition of all operations to ensure the services associated with the visibility relationship: enable/disable a view component and invoke one of CLM interfaces.

It's necessary to check the consistency of the visibility relationship semantic in use of each operation. We emphasize that the provided interfaces by the CLM include the provided interfaces defined by its views components. Use of these interfaces is specified when defining the links of visibility.

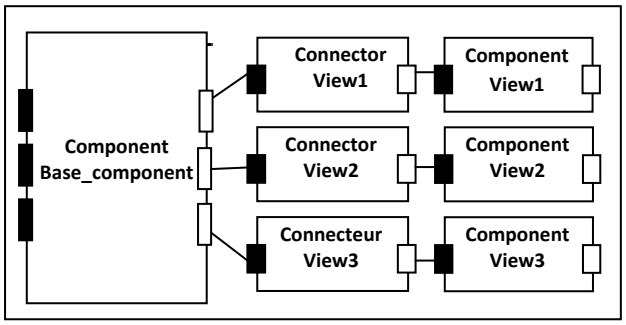

Fig 3: Component model of CLM

4. CLM IMPLEMENTATION IN CCM

4.1 Presentation of the transformation process

According to the MDA approach (Model Driven Architecture) [23] our model of CLM is an independent model of any platform (PIM). From this model, a transformation process has been defined and used for projections CCM [11] and EJB [5][19]. Figure 4 illustrates the transformation process used.

Initially, a new model is automatically generated from the independent supplied model. This new model is a copy of the original model to which is added a set of specific decisions to the target platform that cannot be specified in the independent model.

Once the specific decisions to the target platform are added, another transformation delivers the specific UML model to the chosen platform. This model is a PSM (Platform Specific Model) in the MDA context. The Transformation rules are automatically applied depending on the specific decisions to the target platform.

Finally, the designer can adapt the specific model (PSM) obtained by defining its choices of implementation. These specific models are expressed in using UML profiles. In Figure 4, the specifics items to the CCM platform are expressed in UML [12] and CCM profile (based on CORBA profile [21]. In platform CCM, the assembly is expressed in using XML files, called "assembling descriptors".

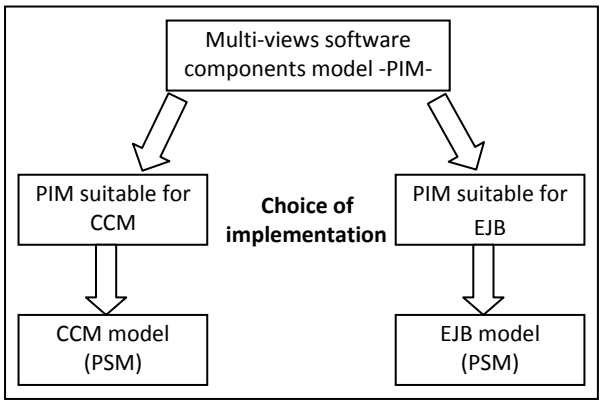

Fig 4: Transformation Process

4.2 The steps of development

In the previous section, we introduced the visibility relationship in CLM model. In this part, we will implement this model for CCM components. To define and use CLM in CCM model, we have developed 3-steps:

4.2.1 Step 1: CLM description - The VIEW- IDL3 language

The CLM description requires the introduction of new concepts. These concepts are added to those defined by the IDL3 language [24]. The CLM description using VIEW-IDL3 defines mainly the following elements: the module's multi-view software component, its base component, its views components and its visibility connectors and their properties.

The following example shows the description of a CLM Media. The CLM declaration in VIEW-IDL3 is just like in IDL3 (Table 1).

***T*able 1. The file of the CLM media declaration in VIEW-IDL3**

```
module demoMedia {
//declaration of interfaces
interface LoansInterface { list of operations….. ; }
interface Counts_AdherentsInterface { list of operations ….. ;
}
interface ExamplariesInterface { list of operations….. ; }
//declaration of  base_component
composant_base Media {
attribute string the_name;
type : Asynchronous ;
port_out LoansInterface ;
port_out ExamplariesInterface ;
port_out Counts_AdherentsInterface ;
port_in LoansInterface ;
port_in ExamplariesInterface ;
port_in Counts_AdherentsInterface ;
view_component  Loans_View {
attribute string the_name;
// Property declaration of visibility relationship
active : false ;
index : 1 ;
dependency : 0 ; // view component Independent of other
                 // components views
mutual_exclusion  : 0 ; //  view component having no mutual
 };                      //exclusion
view_component Examplaries_View {
attribute string the_name;
// Property declaration of visibility relationship
active : false ;
index : 2 ;
dependency : 1 ; // index of view component dependent:
Loans_view: Loans_view
mutual_exclusion : 3 ; //  index of view component having
}                       //mutual exlusion with this view
component
view_component Counts_Adherents_View {
attribute string the_name;
// Property declaration of visibility relationship
active : false;
index :3;
dependency : 1 ; // index of view component dependent:
Loans_view
mutual_exlusion : 3 ; // index of view component having
mutual exlusion with this view component
};
//declaration of  base_component Media home
home MediaHome manages Media { };
// declaration of Loans_View component
component Loans_View {
attribute string the_name;
provides LoansInterface Loans_ViewInterface;
};
//declaration of  Loans_View home
home Loans_ViewHome manages Loans_View { };
// declaration of Counts_Adherents_View component
component Counts_Adherents_View {
attribute string the_name;
provides Counts_AdherentsInterface
Counts_Adherents_ViewInterface;
};
//declaration of  Counts_Adherents_View home
home Counts_Adherents_ViewHome manages
Counts_Adherents_View { };
// declaration of Examplaries_View component
component Examplaries_View {
attribute string the_name;
provides ExamplariesInterface Examplaries_ViewInterface ;
};
//declaration of Examplaries_View home
home Examplaries_ViewHome manages Examplaries_View { };
```

All definitions we have introduced in IDL3 language are summarized in the table below (Table 2).

Table 2. Definitions introduced in IDL3 language

CONCEPT	CONCEPT DESCRIPTION
Base_component	To specify a base component type
View_component	To specify a view component type
index	Each view component has its own index that identifies the other views components.
dependency	If it's different from zero then there is a dependency between two Views components. This value represents the index of the dependent view component. If it's equal to zero so there is no dependency between the current view component and another view component.
mutual_exclusion	If it's not zero then there is a mutual exclusion between two views components. This number represents the index of the view component in mutual

	exclusion. If it's equal to zero then there is no mutual exclusion between the current view component and another view component.
active	Takes the two values: True: the view component is active False: the view component is inactive
port_in	Represents the type of interface required (receptacle or event sink)
port_out	Represents the type of the interface provided (a facet or source event)

4.2.2 Step 2: Projection of the CLM description from VIEW-IDL3 to IDL3

The projection of the CLM description from VIEW-IDL3 to IDL3 by the **View-IDL3ToIDL3** compiler follows well-defined rules (Figure 5). However, the main rule that a type CLM is transformed into a set of components representing the base component, views components and visibility connectors.

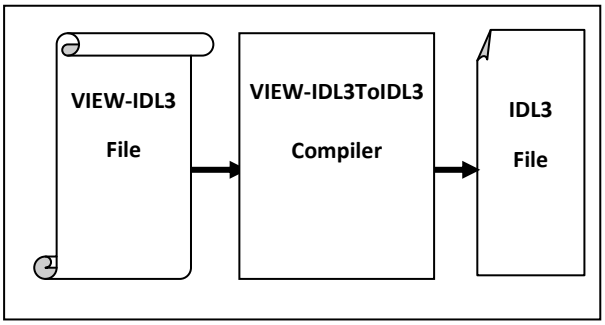

Fig 5: Process projection files VIEW-IDL3 to IDL3

The projection of the CLM description from VIEW-IDL3 to IDL3 product:

1. Definition of a component "component_name" with its home "component_nameHome";
2. Each link description defines a component that has the name of component suffixed by the **Connector** word. The same suffix will be added to its home;
3. Description of the views components and the base component remains the same also their homes declaration;
4. Depending on the value of type: **synchronous**, creates a facet for the base component (if port_in=true) and a receptacle for the view component; **asynchronous** creates an event source for the base component and an event sink for view component;
5. Attributes declaration remains as it's;
6. The ports input/output keep the same number, but they change the way statement.

To illustrate the projection results from the VIEW IDL3 to IDL3, we present the projection of the example in the previous section (Table 3).

Table 3. Projection excerpt from the Media description in IDL3

```
// Declaration of CLM Media in IDL3
module demoMedia {
// Definition of interfaces
interface LoansInterface  {list of operations... ;};
interface Counts_AdherentsInterface
{liste opérations... ;};
interface ExamplariesInterface  {liste opérations… ;};
// Description  of base_component component
component Media_Base_Component {
attribute string the_name;
provides LoansInterface ;
provides ExamplariesInterface ;
provides Counts_AdherentsInterface  ;
uses LoansInterface  To_LoansViews;
uses ExamplariesInterface  To_ExamplariesViews ;
uses Counts_AdherentsInterface
                                To_Counts_AdherentsViews  ;
// Declaration of base_component home
home Media_Base_ComponentHome manages
 Media_Base_Component { };
// Declaration of Loans_View component
component Loans_View {
attribute string the_name;
provides LoansInterface Loans_ViewInterface;
};
// Declaration of Loans_View home
home Loans_ViewHome manages Loans_View { };
// Description of Loans_ViewConnect connector
component Loans_ViewConnect {
attribute string the_name;
// variables Declaration of  Loans_ViewConnect
attribute boolean active ;
attribute int index;
attribute int dependency;
attribute int mutual_exclusion;
// Declaration of Loans_ViewConnect  interfaces
provides LoansInterface for_MediaLoansCon ;
uses LoansInterface to_LoansCon ; };
// Declaration of Loans_ViewConnect home
home Loans_ViewConnectHome manages
                        Loans_ViewConnect { };
```

4.2.3 Step 3: The components interconnection

After implementing the business code of CLM, its base component, its views components and visibility connectors, we completed the interconnection of these components. This should be automated in our model. Indeed, a Java file containing the set of connections between the connectors and the components must be automatically generated. A tool dedicated to this task is under implementation.

5. CONCLUSION

In this article, we presented a model based on multi-view software components. This model CLM combines the advantages of the viewpoint approach, and the advantages of component-based software developing. The CLM model allows to build a software component by assembling a base component and views components. The view component is linked with the base component by a new type of connector called visibility connector.

The proposed model is independent of any platform which conforms to model driven architecture (MDA).

We also presented the multi-view software component implementation on CCM which validated the proposed model and the steps required to design a multi-view software component, beginning with the description until execution on OpenCCM platform.

Thus, it should be noted that the interconnection between the base component with the view components, according to our model, is automatically. The description of a CLM type requires the introduction of new concepts which are added to those defined by the IDL3 language.

6. REFERENCES

[1] Anwar A., Ebersold S., Coulette B., M. Nassar, Kriouile A. 2010. A Rule-Driven Approach for composing Viewpoint-oriented Models. Journal of Object Technology, 89-114.

[2] Finkelstein A., Kramer J., Goedicke M. 1990. *Viewpoint Oriented Software Development*. Presented at the Proceedings of Sciences Engineering and Applications Conference, (1990) Toulouse, France.

[3] Hair A., A UML extension for viewpoint-oriented modeling. Presented at the Proceedings of the International Conference IADIS Appied Compting (2005), Algarve, Portugal.

[4] Hair A. 2004. Analysis and design process based on the viewpoint concept. RITA - Revista de Informática Teorica e Aplicada -, Vol. X, 63-75.

[5] Rubinger A. L., Burke B. 2010. Enterprise JavaBeans 3.1, 6th Edition O'Reilly Media, USA.

[6] Pope A. 2000. The CORBA Reference Guide: Understanding the Common Object Request Broker Architecture, Amazon Edition, France.

[7] Rundensteiner Elke A. 1992. MultiView A Methodology for Supporting Multiple Views in Object-Oriented Databases. Presented at the Proceedings *of the 18th International Conference on Very Large Data Bases*, Vancouver, Canada.

[8] Carré B., Geib J.M. 1991. The Point of View Notion for Multiple Inheritance. Presented at the Proceedings of European Conference on Object-Oriented Programming, Geneva, Switzerland.

[9] Coulette B., Kriouile A., Marcaillou S. 1996. The views approach in object-oriented development of complex systems. Object Review, 13-20.

[10] Traverson B., Yahiaoui N. 2002. *Connector for CORBA Components.* Presented at the Proceedings of the 8th International Conference on Object-Oriented Information Systems, Montpellier, France.

[11] CCM. 2007. Object Management Group, CORBA Component Model Specification OMG Available Specification Version 4.0.

[12] Booch G., Rumbaugh J., Jacobson I. 2005. The Unified Modeling Language User Guide. (2nd Edition) Amazon.

[13] Gottlob G., Schrefl M., Rock B. 1996. Extending Object- Oriented Systems with Roles. ACM Transactions on Information Systems -TOIS-, 268-296.

[14] Kiczales G., Lamping J., Mendhekar A., Maeda C., Videira Lopes C., Loingtier J.M., Irwin J. 1997. Aspect-Oriented Programming. Presented at the Proceedings of European Conference on Object- Oriented Programming, Jyväskylä, Finland.

[15] Ossher H., Kaplan M., Harrison W., Katz A., Kruskal V. 1995. Subject-oriented composition rules. Presented at the Proceedings of Tenth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, *Austin*, Texas.

[16] Mili H., Dargham J. 2000. Views : A Framework for Feature-Based Development and Distribution of OO Applications. Presented at the Proceedings of Thirty-Third Hawaii International Conference on System Sciences, Honolulu, HI.

[17] Krumeich J., Werth D., Loos P. 2013. Towards a Viewpoint-based Modeling Method to Foster Collaborative Modeling Conceptual Design and Implementation. Presented at the Proceedings of the Pacific Asia Conference on Information Systems, Jeju Island, South Korea.

[18] Krumeich J., Werth D., Loos P. 2014. Conceiving a method for viewpoint-based modeling using recommender systems in a multiple-user environment – Conceptual approach and proof-of- concept. Presented at the Proceedings of the Twenty Second European Conference on Information Systems, Tel Aviv.

[19] Wetherbee J. , Rathod C., Kodali R., Zadrozny P. 2015. Beginning EJB 3: Java EE 7 Edition, Kindle Edition, France.

[20] Debauwer L., Caron O., Carré B. 2000. Contextualization of OODB Schemas in CROME. Presented at the Proceedings of the 11th International Conference DEXA, London, UK.

[21] Fernandez L.F., Mareno A. V. 2004. An Introduction to UML profiles. European Journal for the Informatics Professional, Vol. 7, 6-13.

[22] Belloir N., Barbier F. 2004. *Checking a priori model of software.* Presented at the Proceedings of the INFormatique des ORganisations et Systèmes d'Information et de Décision INFORSID), Biarritz, France.

[23] Pastor O., Molina J. C. 2013. Model-Driven Architecture in Practice: A Software Production Environment Based on Conceptual Modeling. Springer.

[24] ACCORD, Projet ACCORD 2003, *Modèle abstrait d'assemblage de composants par contrats*, Livrable-4.

[25] Miles R., Hamilton K. 2013, Learning UML 2.0. O'Reilly Media, USA.

[26] Nikunj R.M., Medidovic N., Phadke S. 2000. *Towards a Taxonomy of Software Connector*s. Presented at the Proceedings of the 22nd International Conference on Software Engineering, Limerick, Ireland.

[27] Marcaillou S., Coulette B., Kriouile A. 1994. Visibility : A new relationship for complex system modeling. TOOLS, Santa Barbara, USA.

[28] Nassar M. 2003. VUML, A Viewpoint oriented UML Extension. Presented at the Proceedings of the 18 th IEEE International Conference on Automated Software Engineering. Montreal, Canada.

[29] Sherpa P. 1995. Project Tropes 1.0 reference manual, INRIA Rhônes-Alpes IMAG-LIFIA, Grenoble, France.