Approches par points de vue pour l'ingénierie des Systèmes d'information

Viewpoint Approaches on Information Systems Engineering

Brahim Lahna

Laboratoire LIG, France et Laboratoire SIR, Ecole Mohammadia d'ingénieurs, Maroc

brahim.lahna@imag.fr

Ounsa Roudiès

Laboratoire SIR, Ecole Mohammadia d'Ingénieurs, Av Ibn Sina, BP 765 Rabat-Agdal, Maroc

roudies@emi.ac.ma

Jean-Pierre Giraudin

Laboratoire LIG, Maison Jean Kuntzmann, 110 avenue de la Chimie, BP 53 – 38041 Grenoble cedex 9, France

giraudin@imag.fr

Résumé

Les concepts de point de vue, de rôle, de perspective et de vue ont été largement étudiés et utilisés dans de nombreux travaux, dans divers domaines de l'informatique. On retrouve ces notions dans les Systèmes de Représentation de Connaissances par Objets, les Bases de Données Orientées Objets, les Méthodes de Conception, la Programmation Orientée Objets, etc. Chaque domaine lui donne sa propre définition et l'utilise dans son propre contexte.

La notion de vue se développe indépendamment dans ces différentes disciplines. Bien que les propositions différent, elles sont confrontées aux mêmes problèmes : établir des relations entre vues et garantir la cohérence des vues entre elles ou vis-à-vis d'un modèle de référence. Nous présentons dans cet article des travaux relatifs aux notions de vues et points de vue dans différentes disciplines informatiques en insistant sur une proposition originale pour une nouvelle ingénierie des systèmes d'information orientée perspectives et composants multivues réutilisables.

Mots-clés

point de vue, vue, perspective, intégration, personnalisation, aspect, rôle, préoccupation, revue électronique, eTI, technologie de l'information

Keywords

viewpoint, view, perspective, integration, personnalization, aspect, role, concern, electronique journal, eTI, information technology

1. Introduction

La complexité des systèmes logiciels ne cesse de croître avec, notamment, l'explosion des besoins et la diversité des contraintes techniques. Par conséquent, leur développement devient de plus en plus complexe, coûteux et difficile et requiert de nouvelles approches méthodologiques. Dans ce contexte, la structuration en vues introduit un mécanisme de décomposition souple, capable de réduire la taille et la complexité d'un problème. L'intérêt de cette notion de vue pour l'ingénierie des systèmes logiciels a été souligné par de nombreux chercheurs dans différents domaines (Shaw *et al.*, 1996) et confirmé en pratique par des industriels (Kruchten, 1995).
La notion de vue est apparue simultanément et indépendamment dans plusieurs domaines tels que les bases de données, les spécifications formelles, les architectures logicielles ou les méthodes de conception. Cette simultanéité se reflète dans la variété des appellations : vue, perspective, aspect, rôle, facette, sujet, préoccupation, *etc.* L'idée commune est d'identifier des utilisations différentes d'un même artefact et de les représenter de manière séparée afin que, dans chaque contexte, on puisse considérer l'artefact d'une manière plus simple et correspondant aux préoccupations de ce contexte. Ces préoccupations se distinguent par une variété de critères tels que le découpage fonctionnel, la catégorie d'utilisateurs, le niveau d'abstraction ou la phase du cycle de vie (Dijkman *et al.*, 2008).
Bien que les propositions diffèrent, elles sont confrontées aux mêmes problèmes : séparer les préoccupations, modéliser les vues et offrir des opérateurs de manipulation ou de composition, établir des relations entre vues et garantir une forme de cohérence des vues entre elles ou vis-à-vis d'un modèle de référence.
Nous présentons dans cet article une revue des travaux relatifs à la notion de points de vue en insistant sur les utilisations, définitions et représentations dans une perspective d'ingénierie des systèmes d'information. Dans les sections deux à six, nous avons distingué les disciplines informatiques suivantes : la représentation des connaissances, les bases de données orientées objets, la programmation par objets, les architectures logicielles et l'ingénierie des exigences. La dernière section commence par la conclusion de cette réflexion, ensuite nous introduisons notre approche originale de prise en compte d'une modélisation multivue, enfin nous dégageons des perspectives à ce travail de recherche.

2. Points de vue en représentation des connaissances

C'est en représentation de connaissances que la notion de point de vue ou perspective a tout d'abord été considérée dans des modèles à objets. Elle réfère à la capacité d'un système à modéliser une même entité selon des points de vue différents, reflétant les diverses perceptions que des concepteurs ou utilisateurs ont de cette entité.
Une des premières références au terme perspective se trouve dans le travail de Minsky (Minsky, 1975) avec une connotation spatiale. Pour lui, un objet peut être vu par des observateurs différents à partir de divers points de vue ; ces observateurs regardent tous les mêmes attributs mais chacun peut les voir avec des valeurs différentes selon ses propres points de vue. Bien que différentes, ces valeurs sont unifiables par des opérations de transformation spatiale (Mariño, 1993).
Dans les sections suivantes nous résumons quatre systèmes qui illustrent la représentation explicite des perspectives dans les représentations des connaissances par objets : KRL, LOOPS, TROPES et ROME.

2.1 KRL

Knowledge Representation Language, KRL, est l'un des premiers langages destiné à représenter des connaissances avec Frames (Masini *et al*., 89) et à reconnaître qu'un objet peut être considéré de plusieurs façons, selon le point de vue de l'observateur. Chaque point de vue est représenté dans un objet séparé, appelé perspective, de sorte que l'objet tout entier apparaisse comme agrégation de toutes ses perspectives. Les perspectives servent dans KRL comme une simulation de la spécialisation multiple. Le modèle de KRL possède un méta niveau qui décrit l'ensemble des concepts du modèle (classe, objet, événement, relation, etc.) par des métaclasses appelées catégories. Trois des sept types de catégories permettent le traitement de perspectives.
Les catégories Basic sont des racines des graphes des différentes familles d'objets. Elles servent à partitionner le monde réel en des familles disjointes d'objets tels que Personne et Ecole, et établissent une première partition de l'univers du discours. Les catégories Abstract sont utilisées pour factoriser des informations à la manière des classes abstraites. Les catégories Specialization spécialisent les catégories Basic ou Specialization. Les catégories **Individuals** décrivent des entités réelles du monde (des individus).
Les perspectives sont des notions centrales à la représentation des connaissances en KRL. Un objet peut être défini par plusieurs perspectives qui correspondent à autant de points de vue selon lesquels il peut être considéré (Bobrow *et al*., 1977). KRL les modélise avec le descripteur Perspective. Un individu a une première Perspective qui est la classe la plus générale à laquelle il appartient, une unité de type Basic et il peut avoir d'autres Perspectives parmi les unités de spécialisation de sa classe de base. Ainsi, en figure 1, la classe de base est la classe Membre qui peut être spécialisée en diverses unités, telles que Professeur, Chercheur, Assistant, Droit, Science, etc. L'unité individuelle, Pierre, est un membre qui, selon la perspective Assistant est un assistant de classe A et, selon la perspective Droit est un enseignant de Droit public.

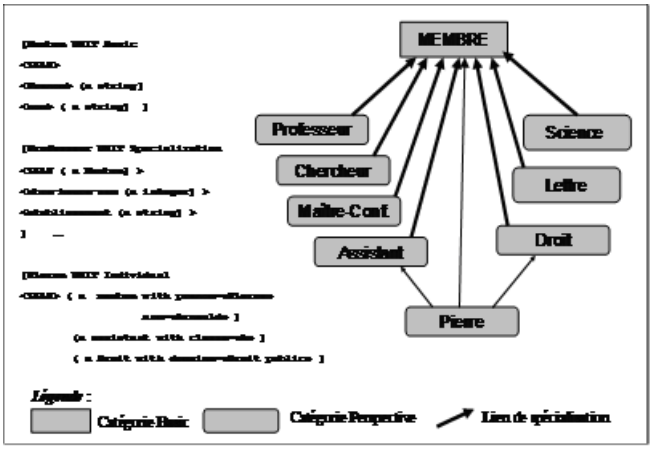

Figure 1. Les perspectives dans KRL

Les perspectives dans KRL sont représentées au niveau des instances. Une instance est liée à son unité de base et à ses perspectives. Les attributs hérités de chacune de ces unités sont organisés dans des groupes d'attributs.
Pour définir une unité individuelle de plusieurs points de vue, on lui associe plusieurs descripteurs de type **Perspective** parmi lesquels un seul possède un prototype de type **Basic** alors que les autres possèdent des prototypes de type **Specialization**. Les attributs sont groupés selon la perspective à partir de laquelle ils décrivent l'objet.

2.2 LOOPS

Alors que dans le modèle de KRL, une perspective d'un objet est un sous-ensemble d'attributs, dans le modèle LOOPS (Stefik *et al*., 1985), une perspective est un objet à part entière qui peut recevoir directement des messages. Pour traiter les perspectives, LOOPS utilise deux classes abstraites (mixin) : Node et Perspective. Une classe qui décrit des objets ayant des points de vue, est une sous-classe de Node, alors qu'une classe décrivant des objets qui sont des perspectives d'autres objets, est une sous-classe de Perspective. En figure 2, la classe Membre est une sous-classe du mixin Node à laquelle on associe des sous-classes de Perspective (Assistant et Droit) qui décrivent les perspectives d'un objet instance de Membre.

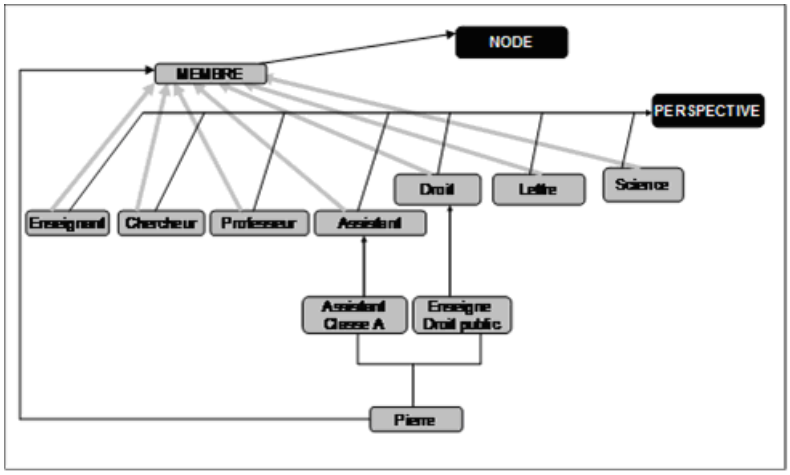

Figure 2. Perspectives d'un objet dans LOOPS

Les perspectives d'un objet LOOPS sont des objets indépendants, instances des sous-classes du mixin Perspective. L'objet même est membre d'une sous-classe du mixin Node. Un objet et ses perspectives sont une sorte d'objet composite, les composants étant les différentes perspectives. Chaque perspective est un objet indépendant pouvant recevoir des messages, ce qui autorise d'avoir des attributs de même nom avec des sens différents dans plusieurs perspectives.
Les perspectives sont toutes liées à l'objet. Le lien entre perspectives est un lien conceptuel. En reprenant l'exemple précédent, l'individu Pierre est lié à ses deux perspectives : Pierre comme un assistant de classe A et Pierre comme un enseignant de droit public. A la différence des objets composites, les perspectives sont créées dynamiquement à la demande. Ainsi, pourrait-on ajouter ensuite une perspective Genre, divisant les membres en deux classes, et considérer Pierre comme Homme.

2.3 TROPES

Tropes est un modèle de représentation des connaissances à objets multipoints de vue. TROPES partitionne la base de connaissances en familles disjointes appelées Concepts. Un Concept peut être vu et manipulé selon des points de vue différents. Chaque point de vue donne lieu à une structuration particulière de la connaissance du Concept dans un graphe de classes. L'aspect pluridisciplinaire d'une base de connaissances se reflète dans l'ensemble de points de vue. Ceux-ci peuvent être reliés par une ou plusieurs passerelles.
Les passerelles permettent de définir les liens sémantiques existant entre deux points de vue d'une même instance et par conséquent de gérer leur cohérence. Une instance satisfaisant les contraintes définies dans un point de vue respecte forcément celles qui ont été définies dans un point de vue faisant partie de la même passerelle. Les passerelles permettent d'établir des relations d'inclusion

ensembliste entre des classes de points de vue différents et reflètent l'aspect interdisciplinaire d'une base de connaissances.
Dans TROPES, un individu du monde est représenté par une instance d'un Concept qui est rattachée à sa classe la plus spécialisée dans chacun des points de vue du Concept. De plus, TROPES permet la manipulation d'objets composites qui sont décomposables différemment selon les points de vue de son concept (Mariño, 1993).
La figure 3 représente la décomposition du concept de Membre en trois points de vue, à savoir Enseignement, Recherche et Statut, et deux passerelles, l'une reliant les racines des hiérarchies des points de vue et l'autre reliant les classes représentant les membres temporaires et les membres à temps partiel.

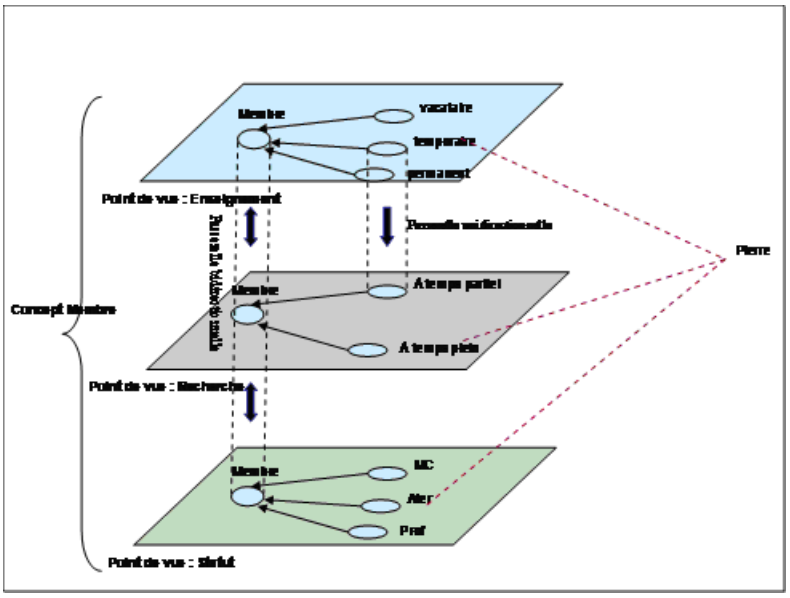

Figure 3. Points de vue en TROPES

Une instance est reliée par un lien « est un » à une classe participant à un point de vue donné et « appartient » (Gensel, 1993) alors à toutes les classes de la hiérarchie d'héritage situées entre la racine de l'arborescence et la classe d'instanciation. Un point de vue définit des masques de visibilité des attributs définis dans le Concept, dans la mesure où les attributs sont définis dans le Concept, mais peuvent aussi avoir des définitions différentes dans les classes. Le Concept TROPES est en fait un ensemble d'instances potentielles ayant des représentations en fonction du point de vue auquel elles appartiennent.

2.4 ROME

Les travaux concernant le projet ROME, « Représentation d'Objet Multiple et Evolutive », (Carré, 1989) et ses évolutions FROME, « Frame en ROME », (Dekker, 1994) et CROME, « Contextes en ROME » (Debrauwer, 1998) (Vanwormhoudt, 1999) traitent de l'intégration des points de vue dans une représentation objet de connaissances. Le concept de représentation multiple et évolutive est proposé pour combler les lacunes des langages orientés objet dans la représentation de l'évolution des objets et de leur appartenance à plusieurs points de vue.
ROME reprend la notion d'instanciation des langages à objets : une classe a des méthodes pour créer des instances (appelées I-CLASSE) qui sont définitivement attachées à cette classe par le lien d'instanciation. Pour représenter les instances, ROME introduit la notion de classe de représentation ou R-CLASSE qui, par opposition à une classe d'instanciation, n'a pas la fonctionnalité d'instanciation. Les classes de représentation sont des spécialisations d'une classe d'instanciation ; elles décrivent des sous-ensembles d'individus ayant des propriétés spécifiques. De cette façon, un objet est instance d'une unique classe et

peut être représentant de plusieurs classes, c'est-à-dire avoir plusieurs points de vue. Ceci constitue la représentation multiple.
L'objet étant lié à plusieurs classes, B. Carré propose une méthode pour faire évoluer cette représentation multiple en rajoutant ou en enlevant des liens de représentation dynamiquement. L'évolution n'affecte pas l'unique lien d'instanciation de l'objet puisqu'elle ne détruit ou ne rajoute que des liens de représentation et permet ainsi de gérer le partage des objets dans le système. La représentation évolutive consiste à faire évoluer les liens de représentation d'un objet.

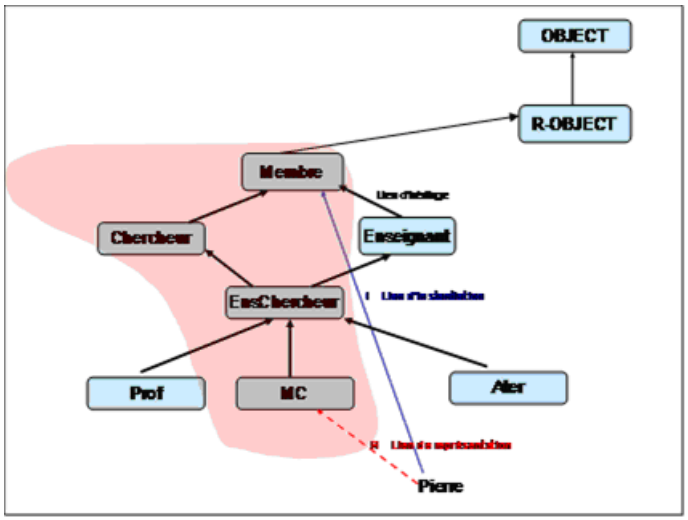

Figure 4. Classes d'instanciation et classes de représentation dans ROME.

Une instance de ROME a une classe fixe d'instanciation mais elle peut avoir plusieurs classes de représentation, sous-classes de sa classe d'instanciation. Les liens de représentation peuvent changer lors de l'évolution de l'instance. La Figure 4 définit la I-Classe Membre et six R-Classes : Prof, Mc, Ater, Enseignant, Chercheur, EnsChercheur. Elles décrivent des r-objets. Soit Pierre un r-objet instance de Membre, le lien d'instanciation qui lie Pierre à Membre est permanent tout au long de l'existence de Pierre. En revanche, Pierre peut évoluer dans le treillis de racine Membre. Ainsi, si Pierre, initialement Ater, devient Maître de conférences, le lien de représentation qui le relie à la classe Ater est remplacé par un autre le reliant à la classe MC. Cette évolution de la représentation est réalisée grâce aux opérations r+ et r- de la classe R-Object. Ainsi, l'opération (Pierre r- Ater) supprime le lien de représentation qui lie Pierre à la classe Ater tandis que l'opération (Pierre r+ MC) instaure un lien de représentation entre Pierre et la classe MC. Cette idée d'avoir une classe d'instanciation pour l'information de base de l'objet et plusieurs classes de représentation pour ses perspectives a été proposée également par le système PINOL (Nguyen *et al.*, 1992) qui distingue le type d'une instance contenant son information structurelle de base, de ses classes qui représentent les différentes perspectives.
ROME offre une autre facilité pour la manipulation des points de vue, à savoir la possibilité de parcourir un sous-graphe du graphe incluant certaines classes spécifiques.

2.5 Bilan

Les approches KRL, LOOPS, TROPES et ROME illustrent différentes solutions apportées à la gestion des points de vue, dans le domaine de la représentation des connaissances. Ces modèles reposent sur l'hypothèse qu'un point de vue est une représentation partielle d'un ensemble cohérent d'objets.
Nous pouvons comparer ces approches en fonction d'un ensemble de caractéristiques :

- La portée d'un point de vue. Dans la plupart des modèles de représentation de connaissances tels que LOOPS, KRL, ROME, une perspective est uniquement

considérée sur un objet. Dans le modèle TROPES, la notion de point de vue porte sur un objet et aussi sur une classe.

- Le type de représentation. Dans le langage KRL, la représentation multiple n'est pas complètement décentralisée. En effet, si les attributs d'un objet sont répartis dans des groupes tels que chaque groupe est relatif à une perspective, il n'est néanmoins pas possible d'accéder à une perspective indépendamment de l'objet. Par contre, dans LOOPS, nous pouvons considérer que la représentation multiple des objets est décentralisée, car une perspective d'un objet est un objet à part entière. Dans le modèle TROPES, la représentation multiple des objets n'est pas décentralisée ; l'état de l'objet contient l'union des attributs décrivant l'entité dans tous les points de vue.

Le tableau ci-dessous, résume les principales caractéristiques des modèles en représentation des connaissances étudiés dans cette section.

Modèle de base		Point de vue		Partage d'attributs entre points de vue	Attributs homonymes dans des points de vue
		entité principale	Mode de définition		
KRL	modèles de frames	objet	représenté par un groupe d'attributs	Non	Non
LOOPS	modèles de frames	objet	représenté par un objet composant	Non	Oui
TROPES	modèles de classes sans héritage multiple	objet	Calculé. Le résultat est un sous ensemble des attributs de l'objet et un graphe d'héritage	Oui	Non
ROME	modèles de classes avec héritage multiple	sur un objet et une classe simultanément	Calculé Le résultat est un graphe d'héritage	Non	Oui

Tableau 1. Comparaison d'approches en représentation de connaissances

3. Vues en bases de données à objets

Le mécanisme de vue dans les bases de données a suscité plusieurs travaux aussi bien dans le contexte relationnel (Barsalou, 1990) (Sheth *et al.*, 1988), (Motro, 1987) que dans le contexte objet (Heiler *et al.*, 1988) (Heiler *et al.*, 1990) (Scholl *et al.*, 1991) (Bertino, 1992).
Dans le modèle relationnel, une vue est une relation virtuelle (non stockée physiquement) définie par une requête sur une ou plusieurs relations stockées ou autres vues. Comme les langages relationnels sont fermés (*i.e.* le résultat d'une requête exprimée dans un langage relationnel est une relation), la relation retournée par une telle requête représente le contenu de la vue. Ainsi, les vues relationnelles peuvent-elle être utilisées (en général) dans n'importe quel contexte dans lequel une relation peut apparaître.
Les vues apportent des facilités d'interrogation, de restructuration et d'intégration de données, tout en permettant l'adaptation des structures de données aux besoins des applications. Elles ont été proposées initialement pour les SGBD relationnels. Dans le contexte orienté objet, outre la restructuration des données, elles permettent l'adaptation du comportement des objets. Les caractéristiques du monde objet sont, bien entendu, intégrées au mécanisme de vues et y jouent un rôle important. Notre étude de l'intégration des concepts de vues dans les bases de données OO est faite à travers les approches O2Views, COCOON, Multiview et Chimera.

3.1 O2Views

O2Views est un outil permettant la définition et l'utilisation des vues dans le système de gestion de bases de données à objets O2 (O2Views, 1995), (Bancilhon *et al.*, 1992). Une fois définie, une vue peut être employée dans l'expression des requêtes et mettre à jour la base de données par des applications. L'outil a été implémenté avec le langage de programmation O2 C, le langage natif d'O2.

Les vues dans O2Views sont définies à trois niveaux : schémas virtuels, classes virtuelles et attributs virtuels (Souza dos Santos *et al.*, 1994) (Souza dos Santos, 1995).
Une vue est un schéma virtuel dérivé d'un autre schéma (réel ou virtuel), appelé le schéma racine de la vue. Les vues peuvent être composées à n'importe quel degré. Une base virtuelle correspond à l'image d'une base réelle (une base instanciée du schéma de base) à travers la vue. La figure 5 illustre la relation entre les entités virtuelles et leurs bases correspondantes pour une dérivation d'une base virtuelle donnée.

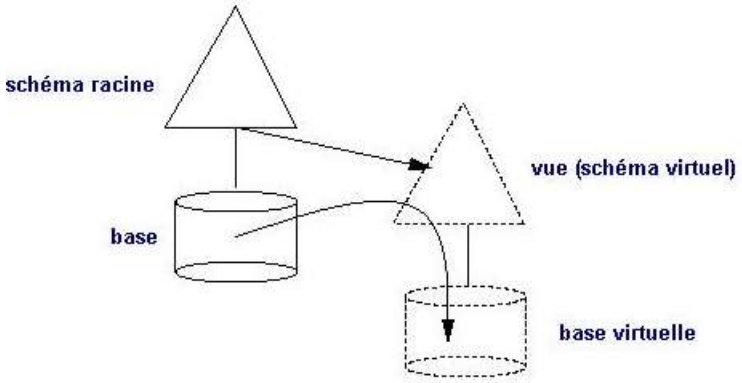

Figure 5. Relation entre schémas et bases dans une dérivation de vues (source O2Views, 1995)

Un schéma virtuel est une collection de classes virtuelles telles qu'un schéma virtuel appliqué à une base (de données) réelle donnera comme résultat une base virtuelle. Les schémas virtuels préservent l'identité de l'objet, c'est-à-dire qu'un objet dans une vue a la même identité que l'objet dans la base réelle. Des schémas virtuels sont dynamiquement activés et désactivés ; l'activation commute le contexte pour la structure et le comportement d'objets d'une base réelle à la base virtuelle dérivée.
Une classe virtuelle est définie par une requête sur la base de données réelle, ou relativement à un type existant par une fonction caractéristique et fournit un ensemble nommé contenant ces objets : l'extension de la classe virtuelle. Ainsi, conceptuellement, l'extension d'une classe virtuelle est définie relativement à l'extension d'une classe de base. D'ailleurs, puisque c'est une classe, son interface peut être modifiée par les primitives de vue pour cacher ou ajouter des attributs (modification de structure) et (re)définir des méthodes (modification de comportement). Les instances de classes virtuelles (objets virtuels) sont les objets présents dans la base de données réelle avec une interface éventuellement différente dans la vue.
Une classe imaginaire choisit et restructure par une requête des données de la base de données réelle ou de la vue et les transforme en de nouveaux objets. L'identifiant d'un objet imaginaire est une fonction de ses attributs de base. Les instances d'une classe imaginaire (objets imaginaires) existent seulement dans la base virtuelle pendant l'activation de la vue.
Un attribut virtuel est une propriété d'une classe virtuelle qui est spécifiée par une fonction de dérivation. Il attache des données à un objet dans la vue, par une requête sur la base de données réelle ou sur la vue. Il augmente l'interface originale des objets (virtuels). Il attache également aux objets imaginaires des données non impliquées par leur identité (attributs différents des attributs de base). Les attributs cachés limitent l'interface des objets virtuels dans la vue et permettent également de cacher des attributs de base des objets imaginaires qui sont censés être utilisés seulement comme clés internes.

3.2 COCOON

Le modèle proposé dans COCOON COCOON, Cocoon Complex-Object-Orientation based on Nested Relations (Scholl et al., 1991) et son langage associé COOL vise l'extension des concepts

du modèle relationnel aux SGBD orientés objet. Le modèle COCOON est basé sur les propriétés objet suivantes :

- L'objet est décrit de façon multiple par plusieurs intensions.
- Les objets ne sont pas encapsulés. L'accès à l'intension des classes est donc libre pour permettre l'expression des requêtes.
- La relation d'héritage est multiple. Une instance peut appartenir à plusieurs classes, même si celles-ci sont indépendantes.
- La classification des instances dans les classes et dans les classes-vues est réalisée de façon automatique et dynamique. Elle est basée sur l'héritage multiple.

COCOON dispose d'opérateurs pour manipuler les types attribués aux instances. Les types peuvent être ajoutés ou retirés dynamiquement de la description d'une instance. La notion de vue dans COCOON est étroitement liée à celle de classe et de requête. Une vue est une classe définie à partir d'une requête portant sur d'autres classes du système. C'est une classe dérivée dont les types des membres et l'extension sont définis implicitement par l'expression de la requête (Scholl *et al.*, 1991). Ainsi, l'extension de la vue est-elle habituellement non stockée explicitement, mais calculée par la requête définissant la vue.
Les vues fournissent une interface spécialisée à quelques objets de base. Un utilisateur ou un programmeur d'applications travaille en général sur une petite partie du schéma global, un sous-schéma, qui est ajusté à ses besoins. Un tel sous-schéma se compose d'une collection de classes de base et/ou de vues ainsi que les fonctions définies sur elles.
Le mécanisme de vue de COCOON permet simplement à des requêtes de servir de définitions de vues, exactement comme dans les SGBD relationnels. Contrairement au modèle relationnel, les vues peuvent être utilisées comme arguments pour des requêtes et des mises à jour, comme les classes ordinaires. Seules quelques restrictions doivent être imposées. En fait, tous les opérateurs de mise à jour ont le même effet que s'ils avaient été appliqués à une classe de base puisque les extensions des vues sont dérivées des classes de base.

3.3 MultiView

MultiView possède un mécanisme de construction de schémas de vues (Rundensteiner, 1992). Un schéma de vues est un ensemble de classes et de classes de vues nécessaires aux utilisateurs de la base de données. Ceux-ci ne perçoivent et ne manipulent ainsi que les informations pertinentes à leur utilisation de la base de données, informations regroupées dans le schéma de vues qui correspond exactement au niveau externe du modèle de l'architecture ANSI/X3/SPARC (Debrauwer, 1998). On peut considérer un schéma de vues comme étant un sous-graphe du schéma global. Plusieurs schémas de vues peuvent être définis sur la même base de données et proposent autant de points de vue différents sur celle-ci.
Dans Multiview, les classes de base sont définies pendant la définition du schéma initial et leurs instances objet sont explicitement stockées en tant qu'objets de base. Par contre, les classes virtuelles sont définies pendant la vie de la base de données en utilisant des requêtes orientées objet, donc leurs définitions sont dynamiquement ajoutées au schéma existant. La classe virtuelle a une fonction associée de dérivation d'adhésion qui déterminera son adhésion basée sur l'état de la base de données. Le contenu d'une classe virtuelle n'est généralement pas explicitement stocké, mais plutôt calculé sur demande. MultiView divise la spécification de vues en trois tâches indépendantes : la personnalisation des classes virutelles, l'intégration de ces classes et la spécification des schémas-vues.
La personnalisation des classes virtuelles est effectuée en utilisant des requêtes orientées objet. Multiview utilise des mécanismes de dérivation de classes pour différents buts tels que pour personnaliser les types, limiter l'accès aux fonctions de propriétés, rassembler des instances d'objets dans des groupes pertinents pour la tâche en cours, etc.

L'intégration des classes virtuelles dans un schéma global cohérent (Rundensteiner, 1992) prend soin de la maintenance de relations explicites entre les classes stokées et les classes dérivées. Ceci est utile pour partager des fonctions de propriétés et des instances d'objets entre classes de façon cohérente, sans duplication inutile. L'intégration des classes assure également la cohérence de toutes les vues avec le schéma global et entre elles.
Quant à la spécification de schémas de vues composés de classes de base et de classes virtuelles, MultiView utilise le schéma global augmenté pour la sélection des classes de base et virtuelles et pour arranger ces classes de vues dans une hiérarchie de classes cohérentes. Ceci supporte la restructuration virtuelle des hiérarchies de généralisation et de décomposition de propriétés, en permettant de cacher et d'exposer des classes dans un schéma-vue.
Cette approche de séparation du processus de conception de vues en tâches bien définies a plusieurs avantages. D'abord, elle simplifie la spécification de vues puisque chacune des tâches peut être résolue indépendamment des autres. En second lieu, elle augmente le niveau du support en tenant compte de l'automatisation de certaines tâches. Des algorithmes automatisent la deuxième et la troisième tâche (Rundensteiner, 1992).

3.4 Chimera

Nous concluons la présentation du concept de vue pour les systèmes de base de données orientées objet par un modèle qui est riche en terme de fonctionnalités mais aussi fondé sur une définition formelle (Guerrini *et al.,* 1994) (Guerrini *et al.,* 1997). Chimera est fondé sur trois modèles de programmation : un modèle de données orienté objet, un langage de requêtes déclaratif basé sur des règles déductives (Bertino *et al.,* 2000) et un langage de règles actives.
Les vues dans Chimera peuvent être des vues préservant l'objet (*object-preserving views*) qui contiennent une sélection d'objets existants maintenant leur identité originale, des vues génératrices d'objets (*object-generating views*) qui créent de nouveaux objets persistants et des vues (set–tuple views) contenant des données dérivées temporaires sans identité persistante.
Comme pour MultiView, les classes de vues introduisent des attributs additionnels non dérivés. À la différence de MultiView, Chimera garde les hiérarchies d'héritage des classes et des vues distinctes. Les deux hiérarchies sont reliées par une hiérarchie de dérivation de vues. La différence entre l'héritage de relations et la dérivation de vues est clairement énoncée : pour l'héritage, le sous-typage de signatures doit concorder avec la formation des sous-ensembles d'extensions. Cette contrainte n'est pas imposée pour la relation de dérivation de vues.
L'héritage multiple et l'instanciation multiple de classes sont supportés. Un objet peut être membre de plusieurs classes plus spécifiques comprenant des classes de vues non reliées par héritage, ce qui facilite la séparation des hiérarchies d'héritage.
Deux niveaux des vues sont conçus : les vues et les schémas de vues. Les vues sont des classes virtuelles et sont utilisables dans n'importe quel contexte où des classes peuvent être utilisées. De même que les classes sont combinées en schémas, les vues sont combinées en schémas de vues. Les schémas de vues permettent de restructurer des schémas pour satisfaire les besoins d'applications spécifiques. Un schéma de vues étant un schéma virtuel, les applications opèrent indifféremment sur des schémas de vues, des vues, des schémas ou des classes.
Pour choisir la facette appropriée d'un objet, Chimera considère le contexte d'une référence. Ainsi, un même objet apparaît-il dans différents rôles, comme instance de plusieurs vues sans retourner aux règles de priorité comme dans O2.

3.5 Bilan

Le mécanisme de vues est un sujet important pour les systèmes de BD orientées objet afin de fournir certaines caractéristiques cruciales au développement d'applications avancées. En raison de la complexité du modèle de données, le paradigme Objet introduit de nouveaux problèmes dans

la définition du mécanisme de vues. Plusieurs approches ont été définies, chacune définissant un mécanisme de vues particulier, conçu pour répondre à un ensemble de fonctionnalités que le mécanisme de vues doit supporter.

	O2	COCOON	MultiView	Chimera
Langage de requêtes	O2	Algèbre objet	Algèbre objet	Règles déductives
Schémas externes	Oui	Non	Oui	Oui
Evolution de schémas	Forme limitée	Forme limitée	Forme limitée	Oui
Problème d'intégration de vues	Hiérarchie vue séparée	Vues insérées dans la hiérarchie de classes	Vues insérées dans la hiérarchie de classes	Hiérarchie vue séparée
Préservation / génération d'objets	Les deux	Préservation	Préservation	Les deux
Existence de schémas de vues	Oui	Non	Oui	Oui

Tableau 2. Tableau comparatif des concepts de points de vue dans les SGBD OO étudiés

Si les vues doivent être utilisées comme les classes, il est essentiel que leurs instances soient des objets et, par conséquent qu'elles aient des identificateurs persistants. En effet, il y a souvent besoin de mettre en référence des instances de vues. Deux sortes distinctes de vues peuvent cependant être identifiées, selon qu'elles préservent ou génèrent l'objet. Les vues préservant l'objet (Object-preserving views) extraient seulement des objets à partir des classes existantes ; les instances de ces vues sont identifiées par les identificateurs des objets de base extraits. Les vues générant l'objet (Object-generating views) créent de nouveaux objets qui doivent donc disposer de nouveaux identifiants.

La plupart des approches (Ohori *et al.*, 1994) (Scholl *et al.*, 1991) (Shilling *et al.*, 1989) considèrent seulement les vues préservant l'objet. Ainsi, les vues fournissent-elles seulement différentes vues d'objets existants. Cette approche est particulièrement utilisée pour supporter des objets avec des interfaces multiples et un comportement dépendant du contexte. Dans ce sens, les vues préservant l'objet sont similaires aux rôles (Albano *et al.*, 1993), (Gottlob *et al.*, 1994), (Richardson *et al.*, 1991). Cependant, ce type de vues n'est pas assez puissant pour supporter tous les types de réorganisation de bases de données.

Comme indiqué dans le tableau 2, O2 et Chimera considèrent les deux types de vues. Ainsi, lorsque l'évaluation de la requête de vues invoque la création d'instances de vues pour peupler la classe-vue (ex. opération de jointure), de nouveaux identifiants sont générés. Par contre, lorsque les vues sont peuplées par l'extraction d'objets d'une classe, éventuellement modifiant leur structure et comportement (ex. opération de sélection ou projection), les instances de vues préservent les identifiants des objets de base.

4. Vues en programmation par objets

L'une des premières approches qui reconnaît explicitement l'existence et le besoin de représenter des vues multiples en chevauchement dans le développement de programmes a été proposée par Rich (1981). L'approche propose de modéliser les programmes et les structures de données en utilisant des « points de vue » multiples et suggère l'utilisation d'un formalisme appelé « *overlay* » pour supporter une telle approche. Un « *overlay* » est un triplet composé de deux plans (chaque plan représentant une vue) et un ensemble de correspondances (égalités logiques) entre ces deux plans. Les correspondances entre plans représentent les relations entre vues et ne sont pas limitées aux programmes et aux structures de données.

En POO, de nombreux modèles de programmation ont exploré le paradigme de point de vue d'un objet. Cette section discute la programmation par vues (Mili *et al.*, 2002), la programmation orientée sujet (Harrison *et al.*, 1993), la programmation orientée aspect (Kiczales *et al.*, 1997), la séparation multidimensionnelle des préoccupations (Ossher *et al.*, 2000), la programmation

structurée en contextes (Vanwormout, 1999) et la programmation avec les objets morcelés (Bardou, 1998).

4.1 Programmation par vues

Dans la programmation par vues (Mili *et al.*, 2002), chaque objet d'une application est appréhendé comme un ensemble de fonctionnalités de base qui sont disponibles, directement ou indirectement, à tous les utilisateurs de l'objet, et un ensemble d'interfaces qui sont spécifiques à des utilisations particulières, et qui peuvent être ajoutées ou retirées pendant l'exécution. Les interfaces correspondent à des types d'utilisateurs ayant des intérêts fonctionnels semblables ou à des utilisateurs ayant différents intérêts fonctionnels. Cette approche fournit un support pour :

- l'accès simultané d'un programme à plusieurs zones ou vues fonctionnelles,
- l'ajout et le retrait des vues (fonctionnelles) pendant l'exécution,
- l'élaboration d'un protocole cohérent et non encombrant d'adressage des objets multivues.

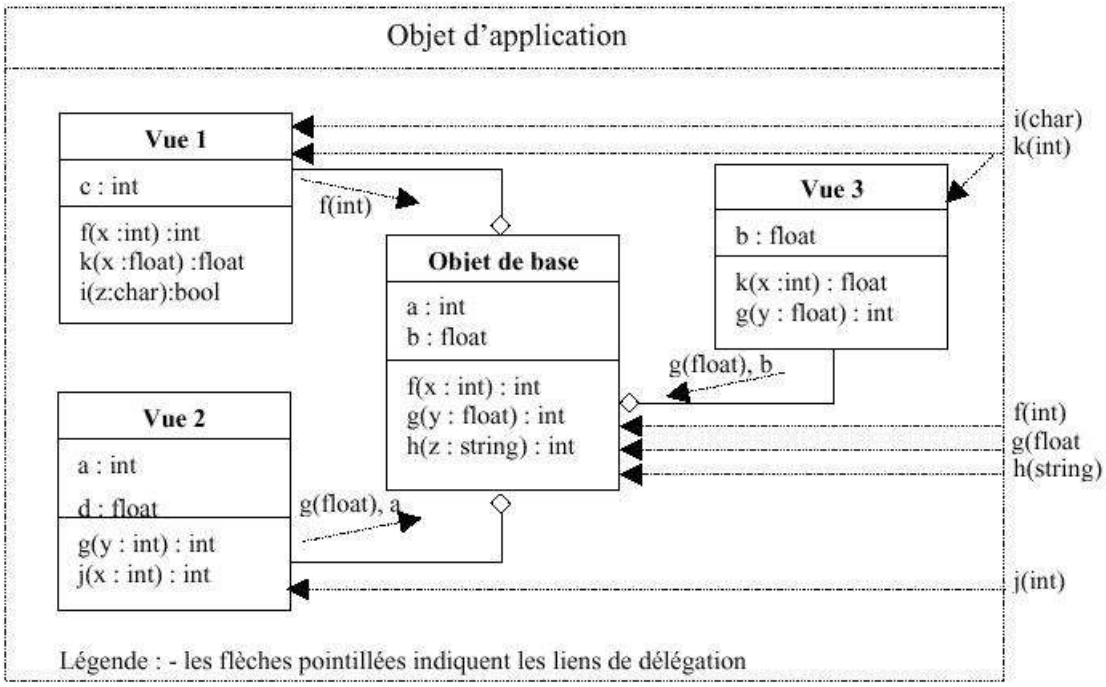

Figure 6. Un modèle objet avec vues (Mili *et al.*, 2002)

La figure 6 montre une implémentation de cette idée basée sur l'agrégation. La bordure en pointillé de l'objet (rectangle) représente l'abstraction d'un objet d'application : c'est la combinaison de l'instance de base et des vues. L'objet de base inclut deux variables d'états ('a' et 'b'), et supporte trois opérations (f(), g() et h()). Les objets vues qui pointent sur l'objet de base peuvent ajouter l'état ('c' pour la vue 1 et 'd ' pour la vue 2), le comportement (i(...) pour la vue 1…), et déléguer les données et comportement partagés. En appelant l'opération f() sur la vue 1, la demande est transmise à l'objet de base, et l'opération f() est exécutée dans le contexte de l'objet de base. Il en est de même pour les références aux variables d'états partagées comme 'a' pour la vue 2. Pratiquement, il y aura une copie simple de telles variables, enregistrée dans l'objet de base, et les demandes de lecture/écriture seront transmises à l'objet de base. L'objet d'application est vu comme supportant l'union des comportements de l'instance de base et des vues actuellement attachées.

Les programmations par rôles ou par points de vue (Kendall, 1999) sont deux paradigmes assez proches. Ils permettent de déstructurer la notion d'objet par l'introduction de vues subjectives : les

rôles qu'ils jouent ou les points de vue qu'ils représentent. Notons par exemple, qu'un système comme Jiazzy (McDirmid *et al.*, 2003) permet de programmer par rôles ou par points de vue.

4.2 Programmation orientée sujets

La programmation orientée sujets (SOP ou *Subject-Oriented Programming*) proposée dans (Harrisson *et al.*, 1993) introduit les termes de subjectivité (*subjectivity*) et sujet (*subject*) dans le paradigme objet et fournit un support pour gérer les points de vue d'un objet.
Un sujet est la spécification d'une hiérarchie de classes. Une même classe peut intervenir dans plusieurs sujets et y définir des variables et des méthodes différentes, reflétant une vision particulière sur l'application. N'étant qu'une abstraction, un sujet ne contient aucune donnée.
Les instances d'un sujet, appelées activations de sujet, contiennent effectivement les données des objets présents dans le système. Un même sujet est activable plusieurs fois et le lien entre ses activations est réalisé au travers de la notion d'identité de l'objet. Un même objet peut être instance de classes différentes dans des sujets différents (Bardou, 1998).
Les activations de sujets sont composées pour produire l'application finale. Cette composition de sujets est exprimée par des règles de composition. Les sujets sont tous vus au même niveau, sans hiérarchie entre eux. Au niveau d'un sujet, les objets sont décrits de façon classique par un ensemble de classes organisées selon la relation d'héritage.
La composition de sujets intervient à des niveaux de détails différents et elle est réalisée de façon semi-automatique. Les règles de bas-niveau spécifient finement les correspondances entre les sujets à composer (Lahire, 2004) (Vanwormhoudt, 1999). Par défaut, la composition utilise l'appariement de noms (*name matching*) pour composer les définitions des classes. Cet appariement peut être remplacé localement par des expressions de composition écrites dans un langage de composition (Ossher *et al.*, 1995).
Un avantage important de la composition de classes dans la SOP est que la fusion des hiérarchies de classes entraîne la propagation de la composition à travers le graphe d'héritage ; c'est-à-dire quand deux classes sont fusionnées, tous leurs descendants tireront bénéfice de cette fusion (dans les deux hiérarchies d'entrée). Une règle de composition implique au moins deux sujets et s'exprime par des opérations simples telles que la fusion, la redéfinition ou le séquencement.
Le paradigme de la POS a évolué pour réduire le couplage entre les sujets, permettant ainsi une meilleure séparation des préoccupations ; c'est la programmation par HyperSpace (Ossher *et al.*, 2000) (HyperJ, 2003).

4.3 Programmation orientée aspects

La programmation orientée aspects (AOP ou *Aspect-Oriented Programming*) (Kiczales *et al.*, 1997), (Kiczales, 2007) est une technique de structuration de programmes permettant la séparation des préoccupations dans les logiciels. Elle se fonde sur une séparation claire entre les préoccupations « métiers » (ou fonctionnelles) et non-fonctionnelles (ou techniques) présentes dans les applications.
La décomposition d'une application fait apparaître :

- un aspect de base qui définit l'ensemble des services (*i.e.* fonctionnalités) réalisés par l'application. Autrement dit, l'aspect de base correspond au "Quoi" de l'application.
- plusieurs aspects complémentaires qui précisent les mécanismes régissant l'exécution de l'application c'est-à-dire les aspects non-fonctionnels définissant le "Comment" (par exemple la synchronisation, la persistance ou la sécurité).

Les aspects sont utilisés pour regrouper des choix d'implémentation qui ont un impact sur l'ensemble du système et qui autrement seraient éparpillés à travers tout le code. Chaque aspect est destiné à être développé de façon indépendante puis intégré à une application par un processus dit de tissage d'aspects (*aspect weaving*). La construction d'une application à partir de différents

aspects nécessite une étape "d'assemblage". En effet, les aspects étant des modules définis séparément les uns des autres, il faut définir leurs règles d'intégration pour les composer afin de "construire" l'application. D'où le besoin d'un mécanisme de composition pour réaliser cet assemblage.
La programmation orientée aspects permet d'encapsuler les préoccupations dans des modules réutilisables (Kiczales *et al.*, 2001). Kiczales *et al.* (1997) distinguent deux types de modules à savoir les composants et les aspects :

- Un composant est un élément encapsulable dans un objet ou dans un module. Les composants sont les unités fonctionnelles d'un système.
- Un aspect n'est pas une unité de décomposition d'un système mais une propriété qui affecte la sémantique des composants d'un système. C'est une fonctionnalité qui n'est pas encapsulée à l'aide des moyens de structuration conventionnels. Les aspects correspondent aux exigences non-fonctionnelles d'un système (Nassar, 2005).

Le but de l'AOP est de séparer la programmation des composants de celle des aspects et de disposer de moyens de tissage pour composer le système final à partir des modules et des règles d'intégration.
L'existence d'un programme de base sur lequel le code aspect est tissé est la différence principale par rapport à l'approche de programmation orientée sujet (Clarke, 2001). Dans cette dernière, il n'y a pas de concept de programme de base, chaque code sujet est indépendant et fournit complètement le code pour l'unité de décomposition particulière qu'il supporte.
Parmi les expérimentations les plus abouties de langages orientés aspects, citons AspectJ (Kiczales *et al.*, 2001), (Lopes *et al.*, 1998), (Kiselev, 2003) développé par l'équipe à l'origine de l'AOP et de JAC (*Java Aspect Component*).

4.4 Séparation multidimensionnelle des préoccupations

C'est un sujet de recherches actif en génie logiciel pendant cette dernière décennie. Elle se rapporte à la capacité d'identifier, encapsuler et manipuler des parties de logiciels qui sont appropriées à un concept, à un but, ou à un motif particulier. Les préoccupations sont les moyens primaires d'organiser et de décomposer le logiciel en plus petites pièces plus faciles à gérer et plus compréhensibles. Plusieurs types de préoccupations peuvent être appropriés à des développeurs dans différents rôles, et pour réaliser différents buts, ou à différentes étapes du cycle de vie du logiciel. Les préoccupations fonctionnelles, métiers, règles de gestion, technologiques sont des exemples de dimensions.
La « séparation multidimensionnelle des préoccupations » implique les points suivants.

- L'existence de multiples types (ou dimensions) de préoccupations.
- La séparation simultanée selon ces multiples préoccupations c'est-à-dire que le développeur n'est pas forcé de choisir un petit nombre (habituellement réduit à un) de dimensions "dominantes" de préoccupations au dépend des autres. Chaque acteur peut donc disposer de sa propre vision d'un système avec ses préoccupations dominantes. Au-delà de l'identification des préoccupations, il est important que l'encapsulation soit suffisante pour limiter l'impact de l'activité lié à l'ascendance d'une préoccupation sur les autres préoccupations. L'absence d'impact entre préoccupations est considérée comme impossible.
- Le chevauchement ou les interactions entre préoccupations. Il est attractif de penser à plusieurs préoccupations comme étant indépendantes ou "orthogonales" mais c'est rarement le cas dans la pratique. Il est important de pouvoir supporter des préoccupations interagissantes les unes sur les autres, tout en offrant toujours la séparation utile.

Un exemple de programmation multidimensionnelle de préoccupations est la programmation par HyperSpace (Ossher *et al.*, 2000), (HyperJ, 2003). Elle se place à la fois comme évolution de la programmation par sujets et une application du paradigme de séparation multidimensionnelle des

préoccupations. Une préoccupation est encapsulée dans un HyperSlice et les règles de composition sont décrites dans des HyperModules. Enfin, une phase de composition nommée intégration réunit tous les comportements pour former l'application finale.
Les trois activités liées à la séparation des préoccupations sont assurées par les HyperSpaces :

- L'identification est la sélection d'une préoccupation, représentée par un HyperSpace. Elle contient les classes et les méthodes qui lui sont pertinentes.
- Les préoccupations sont ensuite encapsulées pour être manipulées comme des classes, dans différents HyperSlices. Un HyperSlice contient une ou plusieurs préoccupations.
- Finalement, pour intégrer correctement les préoccupations, les HyperModules définissent les règles de composition.

4.5 Programmation structurée en contextes

Inspirée de la programmation par sujets, la programmation structurée en contextes vise à supporter la représentation multiple et évolutive d'objets avec points de vue. La motivation principale est d'étudier l'utilisation de référentiels d'objets intervenant dans des contextes applicatifs ou fonctionnels multiples ainsi que de proposer à la fois une décomposition orthogonale en objets de ces systèmes et une décomposition transversale en fonction.
Le projet CROME (Vanwormout, 1999), évolution du projet ROME, représente les objets selon des points de vues capturant les différentes visions propres à chaque acteur du système. En effet, chaque acteur attend une fonctionnalité différente du système. CROME propose un cadre de programmation par objets structurés en contextes fonctionnels, décrits par des plans ou schémas.
Le plan de base contient les éléments communs à plusieurs points de vue. Il définit la structure du référentiel comme une hiérarchie de classes. Les plans fonctionnels adaptent le plan de base à un contexte fonctionnel (une préoccupation particulière) et ajoutent des éléments et des comportements propres à une fonctionnalité. Chaque plan « fonctionnel » définit un point de vue sur les capacités de traitement du système. Ce point de vue est dédié à une activité particulière.
La programmation structurée en contextes sépare les préoccupations par la fourniture de points de vue dédiés à des préoccupations particulières. Contrairement à la programmation orientée aspects, il n'y a pas de notion de tissage statique des préoccupations. La séparation des préoccupations existe toujours à l'exécution dans le référentiel.
Cette approche répond notamment au problème de *crosscutting* entre objets et fonctions. En effet, l'approche objet identifie et découpe le système en entités distinctes, mais ce découpage trop fin ne tient pas forcément compte du découpage fonctionnel du système. Le mécanisme des contextes permet ce découpage du système en fonctions indépendantes.

4.6 Programmation avec les objets morcelés

Les travaux présentés dans (Bardou *et al.*, 1996) et (Malenfant, 1996) sur les objets morcelés dans les langages à prototypes sont issus d'une rationalisation du lien de délégation entre objets existant dans ces langages.
La notion d'objet morcelé vise à donner un statut aux objets en interprétant les morceaux comme des points de vue de celui-ci. Un objet morcelé se présente comme l'agrégation d'entités élémentaires, des morceaux (attributs et méthodes) organisés en hiérarchies de partage d'attributs et de délégation, selon les modalités assez fines offertes par ces langages.
Un morceau appartient toujours à un seul objet morcelé. Il est désigné par un nom unique à l'intérieur de l'objet morcelé et n'a pas d'identité à l'extérieur. Ce nom sert lorsqu'on veut envoyer un message à l'objet selon un point de vue particulier. Dans ce cas, le message est redirigé vers le morceau correspondant ou éventuellement délégué au morceau parent dans la hiérarchie si la méthode n'y est pas trouvée. Des opérations pour ajouter ou supprimer dynamiquement des morceaux à l'objet morcelé sont également proposées. Bardou (1998) aborde l'intégration des objets morcelés dans un langage à classes en respectant les notions sous-jacentes.

4.7 Bilan

Nous avons appréhendé les principaux langages de programmation utilisant le concept de vue. Chacun de ces langages propose sa propre solution et offre de nouveaux concepts et mécanismes. Nous comparons ces propositions selon les trois critères suivants (Tableau 3) :

- Entité principale définit le niveau d'abstraction dans lequel la notion de vue est considérée
- Composition de vues définit la manière dont le partage de propriétés est spécifié dans chacune des approches.
- Niveau d'abstraction définit le niveau d'abstraction de la vue

Approche	Entité principale	Composition de vues	Niveau d'abstraction
Programmation par vues	Classe (des objets vues qui pointent sur un objet de base)	Les classes des différentes vues définissent les objets et les interfaces des classes requises pour la collaboration.	Niveau système
Programmation orientée sujet	Sujet (hiérarchie de classes)	Règles de composition	Niveau système
Programmation par aspects	Aspect et classe	Tissage en utilisant les points de jonction	préoccupation
Séparation multidimensionnelle des préoccupations	Dimension (regroupe un ensemble de préoccupations particulières)	Règles d'intégration	Niveau système
Programmation structurée en contextes	Plan de base et plans fonctionnels	Héritage contextualisé	Niveau système
Programmation avec les objets morcelés	objet	Relation de généralisation/spécialisation	Niveau objet

Tableau 3. Comparaison des concepts de points de vue en programmation OO

5. Méthodes orientées points de vue en ingénierie des exigences

Le but de l'ingénierie des exigences est d'obtenir un ensemble complet et cohérent de spécifications. Ce résultat se construit tout au long du processus ; au départ, les spécifications sont opaques, informelles et s'expriment uniquement par des vues personnelles. Ces vues reflètent les compétences, objectifs et rôles de chaque participant. L'activité de spécification est donc une activité collective. Autoriser l'expression de vues multiples permet notamment une meilleure élicitation des besoins (Nuseibeh *et al*., 1994) (Clarke, 2001) et ne pourra être que bénéfique au processus de spécification.

La notion de points de vue a été abordée dans plusieurs travaux en ingénierie des exigences, en tant que moyen de formulation des spécifications de logiciel (Darke *et al*, 1996) (Kotonya *et al*., 1992) (SE Journal, 1996) (Dubois *et al*., 1988) (Leite et *al.*, 1991). Parmi les approches proposées, nous distinguons celles basées sur des extensions de la méthode SA (*Structured Analysis*), comme la méthode CORE (Mullery, 1979), celles qui utilisent les points de vue en tant que moyen d'intégration de multiples perspectives dans le développement de systèmes (Finkelstein *et al*., 1992) et celles qui utilisent les points de vue comme moyen de définition et de structuration des spécifications (Kotonya *et al*., 1996). Le développement de techniques de méta-point de vue a été également abordé, par exemple dans Preview (Sommerville *et al.*, 1997). Dans la suite, nous détaillerons SADT, CORE, VOSE, VORD.

5.1 SADT

Les points de vue ont été introduits pour la première fois dans la méthode SADT (*Structured Analysis and Design Technique*) développée vers la fin des années 70 par Ross (Lissandre 990) (Marca *et al.*, 1987) (Kotonya *et al.*, 1998).

SADT est une méthode de modélisation des données et des activités d'un système, descendante, modulaire, hiérarchique et structurée. Elle considère que la spécification d'un système est

fortement liée à la façon dont on le perçoit. Elle impose au concepteur de choisir son point de vue dès le départ et de spécifier son système entièrement selon ce point de vue. Plusieurs modèles d'un même système peuvent ainsi être établis selon des points de vue différents. Dans ce cas, les éléments décrits dans les différents modèles SADT sont identiques, mais l'importance de chaque élément, la terminologie utilisée et le niveau de détails nécessaires diffèrent. Le point de vue représente donc la perspective selon laquelle l'ensemble du problème est étudié.
Ainsi, obtient-on des modèles indépendants, chacun fortement lié aux points de vue choisis. Par exemple, « Construire un programme de formation » peut être décrit selon quatre points de vue (responsable pédagogique, formateur, auditeur, directeur de l'organisme de formation) et donner lieu à quatre modèles SADT, puisque les activités sont présentées différemment dans chaque modèle et que les données communes ne se retrouvent pas au même niveau de détails dans chacun des modèles.
Cependant, cette méthode ne propose pas de mécanisme de corrélation entre ces différents modèles, et on obtient une modélisation en multimodèles cloisonnés. De plus, on note une absence de *mapping* des spécifications fonctionnelles à la conception de logiciels. Les spécifications d'analyse et de conception n'ont aucune relation directe et par conséquent la traçabilité est perdue. De meilleures transitions de l'analyse de spécifications à la conception de logiciels rendent les techniques orientées objet plus attrayantes pour les projets industriels de grande taille (Wortmann, 2001).

5.2 CORE

CORE, Controlled Requirements Expression, est la première méthode orientée points de vue (Mullery, 1979) (Mullery, 1985), développée en fin des années 70 pour British Aerospace. Elle est fondée sur la décomposition fonctionnelle et adopte explicitement les points de vue pour formuler les spécifications.
La méthode définit deux types de points de vue : *Defining viewpoints* est relatif aux sous-processus du système, traités selon une approche descendante et *Bounding viewpoints* est relatif aux entités interagissant indirectement avec le système à analyser (Kotonya *et al.*, 1998).
Concept fondamental dans CORE, un point de vue représente une entité physique ou fonctionnelle qui mémorise ou traite l'information. Les fonctions, les sous-parties, les composants d'un système ainsi que son environnement et les opérateurs peuvent tous être modélisés comme des points de vue différents.
La méthode identifie les points de vue le plus tôt et explicitement dans le processus d'élicitation et de spécification des besoins. Les points de vue dans CORE représentent des entités de traitement de l'information qui sont systématiquement analysées lors du déroulement de la méthode. L'orthogonalité des points de vue (non chevauchement) dans CORE est peu réaliste et limitatif si des perspectives multiples d'un système réel doivent être supportées (Nuseibeh, 1994).
La démarche CORE comporte sept étapes itératives et utilise quatre notations différentes de diagrammes avec des annotations textuelles structurées pour définir et analyser les spécifications d'un système. La méthode identifie les activités de chaque étape ainsi que les contrôles à appliquer. Les sept étapes sont : *viewpoint identification, viewpoint structuring, tabular collection, data structuring, single viewpoint modelling, combined viewpoint modelling et constraint analysis.*
Toutes les étapes de CORE utilisent des annotations textuelles structurées. Elles sont développées à l'aide de directives et d'heuristiques. Bien que les étapes soient présentées d'une façon séquentielle, l'analyse des spécifications en utilisant la méthode CORE est un processus itératif.

5.3 VOSE

ViewPoint Oriented System Engineering, VOSE, a été développé à « Imperial College » à Londres, au début des années 90 par Finkelstein, Nuseibeh *et al.* (1992) comme un *framework* pour l'intégration des méthodes de développement dans les systèmes composés (*composite systems*). Ce sont des systèmes qui nécessitent la combinaison de différentes technologies et exigent le travail d'experts dans différents domaines d'application, chacun d'eux ayant son propre intérêt pour le système. Inévitablement ces intérêts se recouvrent, demandant un grand effort pour la gestion et la coordination des spécifications. Cependant, ces domaines de recouvrement ne sont pas facilement identifiables. Une fois que la connaissance dans chaque domaine technologique est représentée de différentes manières, plusieurs stratégies pour le développement sont adoptées. Pour être comparés, les travaux des domaines distincts peuvent ne pas être dans la même étape de développement.
La méthode utilise les points de vue pour diviser les activités et la connaissance des participants concernant le système (Kotonya *et al.*, 1998). Chaque point de vue est défini en tant qu'acteur (ou rôle dans le contexte de développement du système) et son point de vue sur le système, représenté par sa connaissance partielle du système (ses responsabilités). Cette information est organisée dans un point de vue et représentée par un arrangement (*scheme*) de cinq dimensions : *style, domain, specification, work plan et work record.*

- La dimension *Style* décrit le modèle de représentation utilisé par le point de vue.
- La dimension *Work plan* spécifie les actions de développement, le processus et la stratégie du point de vue.
- La dimension *Domain* décrit le sujet de préoccupation du point de vue dans le système global en cours de développement.
- La dimension *Specification* précise le domaine du point de vue dans la notation décrite dans la dimension Style et développée en utilisant la stratégie décrite dans la dimension *Work plan*.
- La dimension *Work record* enregistre l'état de développement et l'historique des spécifications du point de vue. C'est le moyen d'assurer la traçabilité des spécifications.

La dimension *Work Plan* constitue le noyau du point de vue. On y spécifie les actions de contrôle contenant des actions disponibles au développeur pour vérifier la cohérence des spécifications. Les actions de contrôle sont de deux natures : intra ou inter points de vue (*in/out-viewpoint checks*). Les premiers consistent à vérifier la cohérence des spécifications à l'intérieur du point de vue tandis que les seconds contrôlent la cohérence des spécifications d'un point de vue avec celles des autres.

5.4 VORD

VORD a été proposé par Sommerville et Kotonya (1996). C'est une méthode pour la validation préalable des spécifications, principalement dédiée aux systèmes interactifs et à la résolution de points de vue. VORD introduit des points de vue focalisés sur les problèmes utilisateur et les préoccupations d'ordre organisationnel. Le modèle adopté pour les points de vue est orienté service ; les points de vue jouent un rôle analogue aux clients dans un système client-serveur. Les points de vue de VORD sont regroupés en deux classes :
1. Les Points de vue directs correspondent directement aux clients du fait qu'ils reçoivent des services du système et lui envoient des informations de contrôle et des données. Ce sont des opérateurs/utilisateurs du système ou des sous-ensembles connectés au système à analyser.
2. Les points de vue indirects ont un « intérêt » sur une partie ou sur tous les services qui sont fournis par le système mais qui n'interagissent pas directement avec lui. Les points de vue indirects produisent des spécifications qui contraignent les services fournis aux points de vue directs.

Les points de vue sont structurés dans une hiérarchie de classification pour satisfaire les variations des spécifications des utilisateurs. Un point de vue de VORD encapsule un ensemble d'attributs qui aident à définir et structurer les spécifications (Kotonya *et al.*, 1996). Brièvement, un template point de vue comporte les composants suivants :

- *Identifier* attribue un numéro de référence unique à chaque point de vue.
- *Label* et description décrivent le nom et le rôle du point de vue.
- *Type* trace l'origine du point de vue à un point de vue indirect ou direct.
- Attributes caractérisent le point de vue dans le domaine de problème.
- *Requirements* dressent la liste des conditions du point de vue.
- *Event* scenarios décrivent l'interaction entre le point de vue et le système.
- *History* décrit l'évolution des spécifications du point de vue.

Un exemple d'étude de cas pratique plus détaillé est présenté par Kotonya (1999).

5.5 Bilan

Le tableau suivant récapitule la notion de point de vue proposée dans les approches étudiées, Les critères considérés traitent la classification de ces méthodes en ingénierie des exigences et le niveau d'intégration du concept de point de vue dans la méthode.

	SADT	CORE	VOSE	VORD
Notion de pdv	intuitive	faible	défini	défini
Orientation du pdv	Flot de données	processus	Rôle /responsabilité	service
Intégration des exigences fonctionnelles et non fonctionnelles	Non	Non	Non	Oui
Support de la représentation multiple	Oui	Non	Oui	Oui
Support d'événements, spécifications de contrôle	Oui	Oui	Non explicite	Non explicite
Outil support	Oui	Limité	Oui	Oui
Utilisation	Non explicit	Systèmes temps réel	Systèmes composites	Systèmes interactifs
Portée	- Analyse détaillée - Documentation	- Recueil d'information - Documentation	Analyse détaillée	Analyse organisationnelle

Tableau 4. Comparaison des approches étudiées en ingénierie des exigences

6. Frameworks pour la conception multipoint de vue

Dans cette section, nous présentons un aperçu des frameworks dédiés à la conception multipoint de vue. Nous regardons plus particulièrement les relations et la cohérence entre différents points de vue.

6.1 Le Framework GRAAL

Le but du framework GRAAL, *Guidelines Regarding Architecture Alignment*, (Eck *et al.*, 2004) (Wieringa *et al.*, 2003) est d'aligner les parties de conception globale d'un système. À cet effet, il présente les dimensions selon lesquelles les points de vue dans une conception de système peuvent être classifiés. Le framework GRAAL se compose de quatre dimensions orthogonales (aspect, agrégation, raffinement, cycle de vie) illustrées en figure 8. La quatrième dimension (cycle de vie) est dessinée séparément pour surmonter les difficultés d'un schéma quadridimensionnel.

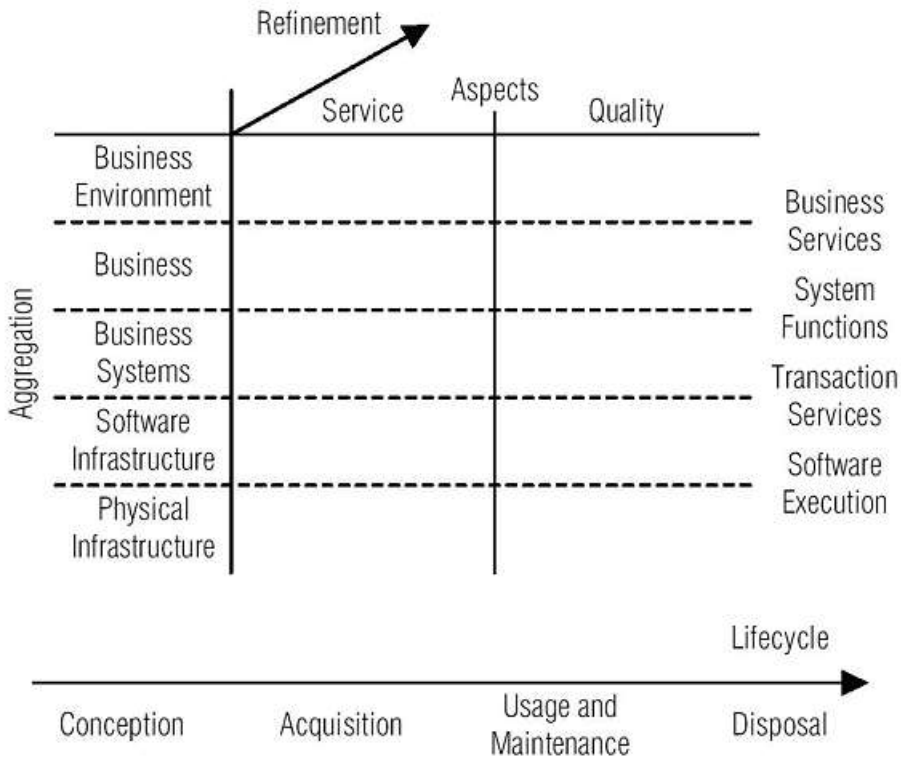

Figure 7. Dimensions dans le Framework GRAAL (Eck et al., 2004)

La dimension «aspects» classe les points de vue selon les propriétés extérieures observables du système traitées par les points de vue. Elle considère qu'un système offre des services avec une certaine qualité et classe les points de vue selon ces deux aspects et plus encore en sous-aspects. Elle aborde des sous-aspects comme l'aspect comportement qui présente les ordres possibles des offres de service et la qualité de service attendue par l'utilisateur.
La dimension «niveau d'agrégation» ordonne les points de vue selon le niveau d'agrégation (des parties du système) considéré par ces points de vue. À chaque niveau d'agrégation quelques parties inter-agissantes du système sont représentées. Ces parties inter-agissantes du système fournissent des services au niveau plus élevé d'agrégation. Parmi les niveaux d'agrégation abordés par le framework, nous citons :

- le niveau d'infrastructure physique qui se compose des parties physiques du système (par exemple PC, réseau) et fournit des services qui permettent à des niveaux plus élevés d'agrégation de s'exécuter ;
- le niveau métier composé des parties métiers (par exemple acteurs, rôles) qui fournit des services métier à un environnement de clients.

La dimension «raffinement» ordonne les points de vue selon le niveau de détail avec lequel les points de vue décrivent le système. L'ajout de détail à un point de vue revient à raffiner ce point de vue. Le framework ne considère pas la décomposition comme une forme de raffinement, parce que la dimension agrégation traite la décomposition d'un système en parties. En revanche, chaque partie du système peut être raffinée indépendamment, en la décrivant avec plus d'informations. Par exemple, nous pouvons décrire chaque partie en décrivant son but et nous pouvons la raffiner en décrivant comment la partie réalise son but (Eck et *al.*, 2004).
La dimension «cycle de vie» ordonne les points de vue selon l'étape du cycle de vie du système abordé par les points de vue. Cette dimension considère les étapes comme : la conception du système, l'utilisation et la maintenance du système.

6.2 ArchiMate

Le but d'ArchiMate (Lankhorst, 2005) est de décrire les relations entre différents points de vue, appelé aussi domaines dans ArchiMate (Lankhorst, 2005) dans une conception de système. À cet effet il présente les concepts qui peuvent être utilisés dans la conception à partir de ces différents points de vue. Il présente également les relations qui peuvent être utilisées pour relier (des instances de) des concepts du même ou de différents points de vue. ArchiMate classe les points de vue par catégorie selon le niveau (métier, application ou technologie) et les aspects qu'ils abordent. La Figure 8 montre les différents aspects et couches dans la conception de système avec ArchiMate.

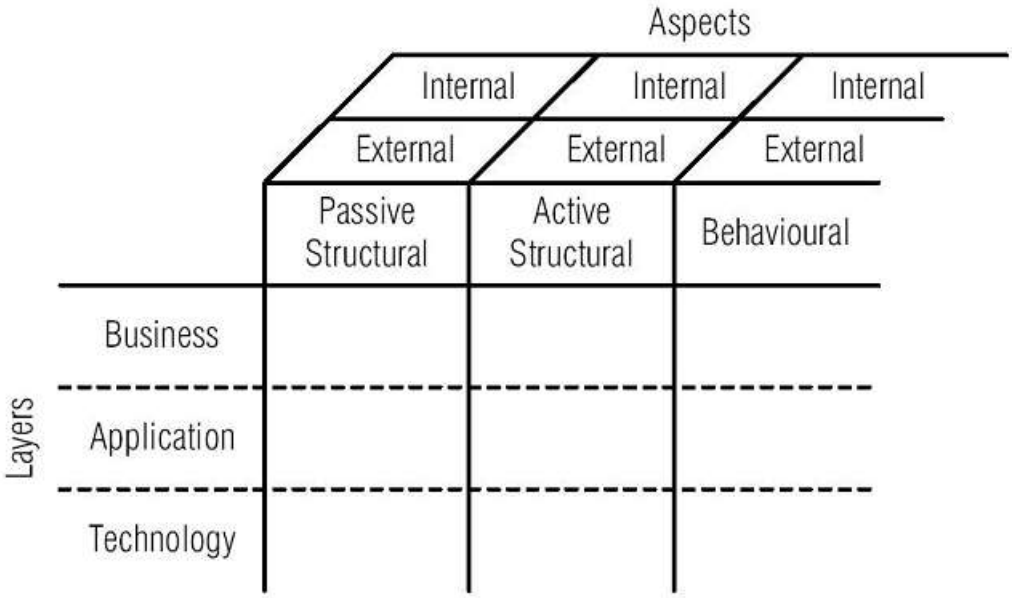

Figure 8. Les couches et les aspects dans ArchiMate (Lankhorst, 2005)

Semblable au framework GRAAL, ArchiMate distingue des niveaux d'agrégation, appelés couches, telles que chaque couche fournit des services aux couches supérieures. Cette architecture distingue trois couches : métier, application et technologie.
ArchiMate distingue les aspects abordés dans chaque couche. Il distingue les aspects structuraux et comportementaux. Les aspects structuraux sont aussi classifiés dans des aspects structuraux actifs et passifs. L'aspect structural actif représente les parties qui prennent l'initiative dans l'exécution des activités (par exemple des acteurs métier ou des composants logiciel actifs). L'aspect structural passif représente les parties qui ne prennent pas d'initiative (par exemple des objets d'information). Dans les aspects structuraux et comportementaux nous pouvons distinguer des aspects externes et internes. Les aspects externes représentent les aspects qui sont observables par les utilisateurs d'une couche. Les aspects internes représentent les aspects qui ne le sont pas.
ArchiMate présente des concepts abstraits et des relations (Lankhorst, 2005) utilisés pour construire une conception en couches. Les concepts abstraits et les relations ne peuvent être utilisés que pour la conception à un haut niveau d'abstraction. Pour construire des conceptions plus détaillées, ArchiMate se base sur des langages spécifiques aux points de vue. Ceci est en conformité avec la philosophie d'ArchiMate, dans laquelle les participants utilisent leurs propres langages et outils afin de construire des conceptions, selon leurs points de vue. Les conceptions des différents points de vue peuvent être importées dans ArchiMate à un haut niveau d'abstraction, abrégeant en conséquence des détails qui sont représentés dans les langages spécifiques de points de vue. Par exemple, une conception de processus métier peut être importée dans ArchiMate. Le résultat est une conception qui représente des activités et un flux de relations entre ces activités, mais sans détails sur le flux de relations.

Nous pouvons utiliser ArchiMate pour relier les activités de processus métier aux services d'application qui les supportent, qui sont importés d'autres langages et outils.

6.3 RM-ODP

Le modèle de référence RM-ODP, *Reference Model for Open Distributed Processing*, (ISO, 2008), présente un modèle de référence qui permet de définir des normes pour la conception et le développement des systèmes distribués ouverts (ODP). Il consiste en un ensemble de concepts et fonctions qui peuvent être utilisés pour définir des normes conformes à RM-ODP.
RM-ODP définit de manière abstraite, et donc adaptable à un contexte d'application, une architecture qui supporte les aspects distribution, réseaux, interopérabilité, portabilité. Ce modèle doit permettre d'organiser en un tout cohérent, les différentes parties d'un système, c'est-à-dire, la portabilité d'une application sur des plates-formes hétérogènes, l'échange d'informations et l'utilisation de fonctionnalités hébergées et offertes par différents systèmes ouverts dans un environnement distribué, et une transparence de la distribution des applications et utilisateurs.
La spécification complète de tout système réparti complexe implique une très grande quantité d'information. Pour bien appréhender cette complexité, le système est considéré à partir de différents points de vue, chacun reflète un ensemble particulier de préoccupations. De cette façon une séparation normale des préoccupations est obtenue (Putman, 2000).
RM-ODP propose cinq points de vue pour spécifier des systèmes ODP, chacun se concentrant sur un sujet de préoccupation particulier. Selon l'ISO, les points de vue RM-ODP forment un ensemble nécessaire et suffisant pour satisfaire les besoins de standards de systèmes répartis ODP. Les points de vue peuvent être appliqués, à un niveau approprié d'abstraction, pour la spécification d'un système complet ODP ou pour la spécification de différents composants d'un système ODP.
Chaque point de vue a un langage de points de vue associé définissant des concepts et des règles pour spécifier des systèmes ODP à partir des point de vue correspondants. Les langages de points de vue sont des langages abstraits dans le sens qu'ils n'impliquent aucune syntaxe ou notation particulière. En principe, n'importe quel langage existant peut être utilisé pour la spécification d'un système d'un point de vue particulier, à condition que ces spécifications puissent être interprétées en termes de concepts du langage de points de vue et adhère aux règles définies (Putman, 2000).
Les concepts de RM-ODP sont structurés selon cinq points de vue :

- Le point de vue entreprise qui définit des concepts et des relations entre concepts pour spécifier le rôle d'un système ODP dans un environnement ;
- Le point de vue traitement qui définit des concepts et des relations entre concepts pour spécifier une décomposition fonctionnelle d'un système ODP ;
- Le point de vue information qui définit des concepts et des relations entre concepts pour spécifier la structure d'information dans un système ODP et les opérations de base qui peuvent être appliquées sur cette information ;
- Le point de vue ingénierie qui définit des concepts et des relations entre concepts pour spécifier les mécanismes et les fonctions qui supportent les interactions distribuées entre les parties du système ODP ;
- Le point de vue technologie qui définit des concepts et des relations entre concepts pour spécifier la technologie sur laquelle un système ODP est implémenté.

La spécification complète du système est alors constituée de l'ensemble des spécifications établies dans chaque point de vue avec leurs règles de correspondance. RM-ODP définit des règles de cohérence qui aident à garder les spécifications de points de vue cohérentes. Les règles de cohérence consistent en des correspondances qui spécifient à quels concepts d'un point de vue correspondent des concepts d'un autre point de vue. Si les concepts d'un point de vue correspondent aux concepts d'un autre point de vue, alors les spécifications de ces points de vue doivent avoir des relations de correspondances. Les relations de correspondances spécifient ainsi les correspondances entre des instances de concepts d'une spécification d'un point de vue et des instances de concepts d'une spécification d'un autre point de vue.

6.4 ViewPoints Framework

ViewPoints Framework (Finkelstein *et al.*, 1994) est l'un des premiers travaux les plus influents sur la conception et la cohérence multipoints de vue. C'est le premier travail qui a changé l'idée des vues en tant qu'abstractions d'informations existantes aux vues qui sont développées séparément par différents *stackeholders*.
Dans *ViewPoints Framework*, chaque point de vue est défini par cinq éléments :

- un domaine qui indique l'univers de discours du point de vue ;
- un style qui définit le langage de modélisation utilisé pour la conception du point de vue ;
- un plan de travail qui définit le processus selon lequel une conception peut être construite à partir du point de vue ;
- une spécification qui représente la conception du point de vue, en utilisant le langage de modélisation défini par le style ;
- un enregistrement de travail qui représente les informations sur l'état courant et l'historique d'une spécification.

La conception complète du système consiste en un ensemble de points de vue et des règles qui définissent des conditions pour la cohérence entre ces points de vue. Chaque règle définit une condition qui doit être maintenue dans une spécification cohérente et définit une action qui doit être prise si cette condition est violée. Les règles sont spécifiées comme des requêtes sur la base de données qui contient la conception.
Pour vérifier la cohérence, *ViewPoints Framework* suppose que les spécifications de points de vue et les règles de cohérence sont définies dans une base de données qui supporte le raisonnement avec la logique de premier ordre. La base de données exécutera alors le contrôle de cohérence.

6.5 OpenViews

OpenViews (Boiten *et al.*, 2000) présente une approche pour maintenir la cohérence dans des conceptions basées sur RM-ODP, bien qu'il puisse être appliqué dans un contexte plus général.
OpenViews définit que deux vues sont cohérentes si on peut trouver une conception qui est un raffinement des deux vues. Les vues peuvent être modélisées en utilisant différents langages de modélisation, dans ce cas l'une des vues doit être transformée, de telle sorte que les vues soient représentées dans le même langage. Plus spécifiquement, OpenViews décrit en détail comment associer les vues représentées dans Lotos (Eijk *et al.*, 1989) et Object-Z (Smith, 2000). Pour ce but, (Derrick *et al.*, 1999) définissent comment transformer une vue représentée dans Lotos en vue représentée dans Object-Z.
OpenViews diffère des approches précédentes qui se fondent sur des relations entre vues et des règles de cohérence. Il diffère de ces approches parce qu'il définit quand deux vues sont dites cohérentes (*i.e.* quand un raffinement cohérent commun peut être trouvé). Les approches ci-dessus définissent que deux vues ne sont pas cohérentes si les règles de cohérences ne sont pas satisfaites.

6.6 Bilan

Le Tableau 5 illustre les niveaux de support que les *frameworks* étudiés retiennent pour définir des relations entre vues et pour contrôler la cohérence entre ces vues. Le tableau classe le support selon deux critères : l'expressivité des relations entre vues et le support conceptuel.

	Expressivité des relations			
		Relations	Directives	Règles de cohérences
Support conceptuel	Aucun			ViewPoints OpenViews
	Concepts abstraits	ArchiMate		
	Concepts abstraits communs		GRAAL	
	Concepts de base communs		RM-ODP	

Tableau 5. Comparaison de *Frameworks* Multipoints de vue

Le critère d'expressivité distingue les *frameworks* selon le support qu'ils ont pour définir des règles de cohérence. Au plus bas niveau, un *framework* supporte la définition des relations entre vues et ne définit pas les règles de cohérence qui s'appliquent à ces relations. Au niveau suivant, le *framework* supporte la définition des règles de cohérence. Cependant, des règles avancées de cohérence telles que le raffinement, ne sont pas supportées. Au niveau le plus élevé, le *framework* supporte pleinement la définition des règles de cohérence.
Un *framework* supporte conceptuellement des relations entre vues, s'il fournit des concepts pour les vues considérées dans le *framework*. Ces concepts fournissent un cadre de référence qui aide les concepteurs à penser aux relations entre vues. D'abord, les concepteurs doivent définir les relations entre les concepts spécifiques aux points de vue et les concepts du *framework*. En second lieu, ils doivent définir la relation entre concepts de différents points de vue, en utilisant les relations entre les concepts du *framework*. Un *framework* peut fournir le support conceptuel à un degré variable.
Les concepts abstraits fournissent des abstractions des concepts qui peuvent être utilisés dans chacune des vues (couvertes par le *framework*). Puisqu'ils sont des abstractions, ils ne peuvent pas couvrir toutes les propriétés de conception en détail. Les concepts abstraits communs ont la propriété supplémentaire qu'ils sont partagés entre vues, là où les concepts abstraits standard sont différents pour chacune des vues. Les concepts de base (communs) couvrent toutes les propriétés de conception en détail.

7. Conclusion : Vers une approche de conception par points de vue

Le logiciel est devenu une partie complexe, sensible et essentielle dans la réussite de la majorité des projets. Pour maîtriser le développement de systèmes de qualité et faire face à l'explosion des besoins en logiciels, il est nécessaire d'élaborer des méthodes de conception adaptées à la complexité des logiciels et à la diversité des contraintes techniques à prendre en compte, le tout avec une exigence de cohérence et un objectif de meilleure adéquation aux différentes perceptions des services à offrir.

7.1 Bilan

Dans cet article, nous avons passé en revue, plusieurs disciplines en informatique qui utilisent les notions de vue, point de vue ou perspective : en représentation de connaissances, en bases de données, en programmation par objets, en phase d'ingénierie des exigences et en architecture de systèmes. L'attrait du recours au concept de vue est double.
D'une part, chaque vue peut être décrite dans un formalisme dédié, adapté au domaine concerné et généralement basé sur une notation graphique intuitive. Le choix de la notation associée à une vue est guidé par les soucis de clarté, de concision et privilégie les abstractions utiles à chaque domaine. D'un point de vue méthodologique, la séparation en vues et les formalismes dédiés simplifient les descriptions, facilitent la compréhension et la communication entre les membres d'un projet.
D'autre part, la décomposition en vues favorise la séparation des problèmes puisque chaque vue se concentre sur un aspect du système. Ce principe de décomposition fournit un puissant mécanisme de structuration.
Partant de cet état de l'art et de ces constats, nous avons élaboré un ensemble de propositions pour développer une démarche d'ingénierie dirigée par des points de vue et s'appuyant sur des composants multivues réutilisables.

7.2 Proposition

L'approche *ViewPoint Oriented Design* (VPOD) que nous préconisons (Lahna *et al.,* 2005a), s'inscrit dans le cadre de la méthode de conception par points de vue COPV (Conception Orientée Points de vues) (Lahna *et al.,* 2005b). Elle propose une démarche d'ingénierie adoptant le principe de réutilisation et d'adaptation de composants multivues, offrant la possibilité de voir le modèle d'un système selon différentes facettes, reflétant les visions des différents acteurs du système. Plus précisément, VPOD propose une extension du métamodèle UML pour guider les concepteurs dans la modélisation de système, en intégrant le concept de points de vues et propose des fragments de démarche sous forme de *patterns* processus et de carte de processus.
VPOD permet d'allier le domaine de conception basé sur les *patterns* et les approches par points de vue afin d'élaborer des composants multivues adaptables et réutilisables et d'en proposer un formalisme de représentation pour aider les concepteurs de SI dans leur tâche de modélisation, en favorisant la réutilisation des composants quelque soit le métier.
L'objectif est d'aider à la réutilisation et à la capitalisation de connaissances afin d'améliorer et d'aider les concepteurs dans leur activité de modélisation de systèmes complexes selon une approche d'intégration de vues. En d'autres termes, VPOD permet de définir un cadre conceptuel de définition d'une démarche d'ingénierie de systèmes d'information à base de composants multivues.

7.3 Perspectives

Actuellement, l'approche VPOD est basée sur des contraintes OCL pour représenter les règles de cohérences. Ces contraintes OCL informent le concepteur quand une règle de cohérence n'est pas respectée. Cependant, elles ne fournissent pas les raisons pour lesquelles cette règle de cohérence est violée. Une telle information est utile pour résoudre les incohérences. Nous proposons le développement de règles de cohérence qui fournissent une telle information. On peut s'inspirer de travaux qui abordent les aspects de traçabilité.
D'un autre côté, nos propositions se sont limitées à l'aspect structurel du système à concevoir. Nous n'avons pas défini les concepts pour représenter la dynamique du système (en termes d'extension du métamodèle UML). Une telle recherche nécessite de s'appuyer non seulement sur des travaux de conceptualisation pour proposer de nouveaux modèles, mais aussi sur des réalisations d'environnements de conception indispensables pour opérationaliser les propositions et en faciliter la validation sur des cas significatifs.

Références

Albano, A. Bergamini, R. Ghelli, G. Orsini. R. (1993). An Object Data Model with Roles. In *Proceedings of the 19th International Conference On Very Large Data Bases*, 39-51.

Bancilhon, F. Delobel, C. Kannelakis, P. (1992). *Building an abject-Oriented Database System- The Story of O2*. Morgan Kaufmann.

Bardou, D. Dony, C. (1996). Split Objects : a Disciplined use of Delegation within Objects, in *ACM SIGPLAN. Proceedings of OOPSLA'96.* San Jose, USA:122-137.

Bardou, D. (1998) *Étude de langages à prototypes, du mécanisme de délégation et de son rapport à la notion de point de vue*. Thèse de Doctorat, LIRMM, Université de Montpellier 2.

Barsalou, T. (1990). *View objects for relational databases*. Thèse de PhD, Computer Science Department- Standford University.

Bertino, E. Guerrini, G. Montesi, D. (2000). Inheritance in a Deductive Object Database Language with Updates, in *Lecture Notes in Computer Science*, vol 1773, 67-85.

Bertino, E. (1992). A view mechanism for object-oriented databases, in *Lecture Notes in Computer Science*, Vienne, Autriche: 580, 136–151, Springer-Verlag.

Bobrow, D.G. Goldstein, P. (1980). Description for a Programming Environment, in *Proceedings of the AAAI*, Stanford University, 187-189.

Boiten, E.A Bowman, H. Derrick, J. Linington, P.F. Steen, M.W. (2000) . Viewpoint consistency in ODP. In *Computer Networks*, 34(3), 503-537.

Carré, B. (1989). *Méthodologie orientée objet pour la représentation des connaissances. Concepts de point de vue, de représentation multiple et évolutive d'objets*, Thèse de Doctorat, Lille.

Clarke, S. (2001). *Composition of Object-Oriented Software Design Models*, Thèse de PhD, Dublin City University.

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

Dekker, L. (1994). *FROME : Représentation multiple et classification d'objets avec points de vue,* Thèse de Doctorat, Université de Lille, Juin 1994.

Derrick, J. Boiten, E.A. Bowman, H. Steen, M.W.A. (1999). Viewpoints and consistency : Translating LOTOS to object-Z. In *Computer Standards and Interfaces*, 21, 251-272.

Dijkman, R.M. Dick Quartel, A.C. van Sinderen, M. (2008). Consistency in multi-viewpoint design of enterprise information systems. in *Information & Software Technology*, 50(7-8), 737-752.

Dubois, E. Hagelstein, J. Rifaut, A. (1988). Formal Requirements Engineering with ERAE in *Philips Journal of Research*, (43) 393-414,.

Eck, P. Blanken, H. M. Wieringa, R. J. (2004). Project GRAAL: Towards operational architecture alignment in *International Journal of Cooperative Information Systems*, 13(3), 235-255.

Eijk, P. Vissers, C. Diaz, M. (1989). *The formal description technique LOTOS*. North-Holland.

Finkelstein, A. Kramer, J. Nuseibeh B., Finkelstein, L. Goedicke, M. (1992). Viewpoints : A Framework for Integrating Multiple Perspectives in *System Development , International Journal of Software Engineering and Knowledge Engineering*, 2(1): 31-58.

Finkelstein, A. Gabbay, D. Hunter, A. Kramer, J. Nuseibeh, B. (1994). Inconsistency handling in multi-perspective specifications in *IEEE Transactions on Software Engineering*, 20(8), 569-578.

Gensel, J. (1993). Expression et satisfaction de contraintes dans TROPES, in *actes de RPO*, La Grande Motte, 51-62.

Gottlob, G. Schrefl, M. Rock, B. (1994). Extending Object-Oriented Systems with Roles in *ACM Transactions on Information Systems*.

Guerrini, G. Bertino, E. Bal, R. (1994). A Formal Definition of the Chimera Object-Oriented Data Model in *Technical Report IDEA*, ESPRIT Project 6333.

Guerrini, G. Bertino, E. Catania, B. Garcia-Molina, J. (1997). A formal model of views for object–oriented database systems in *Theory and Practice of Object Systems*, 3(3):157–183, 250, 251.

Harrison, W. Ossher, H. (1993). Subject-Oriented Programming (A Critique of Pure Objects) in *Proceedings of OOPSLA'93, ACM Press SIGPLAN,* 411-428.

Heiler, S. Zdonik, S.B. (1988). Views, data abstractions and inheritance in *The fugue data model*, pages 225–241, Heidelberg. LNCS 334 Springer-Verlag.

Heiler, S. Zdonik, S.B. (1990). Objects views : Extending the vision in *Proceeding IEEE data Engineering Conf*, USA, Los Angeles: 86–93.

Hyper/J team (2003). Hyper/J. http ://www.research.ibm.com/hyperspace/HyperJ/-HyperJ.htm.

ISO/IEC (2008). *RM-ODP. Reference Model for Open Distributed Processing – Amendment to Parts 2 and 3.* ISO and ITU-T, Geneve, Suisse.

E. Kendall, (1999) Role model designs and implementations with aspect-oriented programming in *Proceedings of OOPSLA'99*. 353–369, ACM Press.

Kiczales, G. Lamping, J. Menhdhekar, A. Maeda, C. Lopes, C. Loingtier, J.-M. Irwin. J. (1997). Aspect-Oriented Programming in *Procceeding of ECOOP'97*. 1241, 220-242. Springer-Verlag.
Kiczales, G. Hilsdale, E. Hugunin, J. Kersten, M. Palm, J. Griswold, W. (2001). *An Overview of AspectJ. AspectJ white paper* submitted to the the European Conference on Object-Oriented Programming (ECOOP'01).
Kiczales, G. (2007). *Making the Code Look Like the Design - Aspects and Other Recent Work*. ICPC
Kiselev, I. (2003). *Aspect-Oriented Programming with AspectJ*. Sams Publisching.
Kotonya, G. Sommerville, I. (1996). Requirements engineering with viewpoints. in *Software Engineering Journal*. 11(1), 5–11.
Kotonya, G. Sommerville, I. (1998). *Requirements Engineering, Processes and Techniques*. Willey
Kotonya, G. (1999) Practical Experience with Viewpoint-Oriented Requirements Specification. *Requirements E*ngineerings. 4(3), 115-133
Kruchten, P. (1995). The 4+1 View Model of Architecture. *IEEE Software*. 12(6),42-50
Lahire, P. (2004) *Habilitation à diriger des recherches*, Université de Nice-Sophia Antipolis - Laboratoire I3S, décembre 2004
Lahna, B. Roudiès, O. Giraudin, J-P. (2005a). Using UML Extensions to Model Multiview Component-Based Systems. *Proceeding of ICICIS'2005, Second International Conference on Intelligent Computing and Information Systems*. Le Caire, Egypte.
Lahna, B. Roudiès, O. Giraudin, J-P. (2005b). Une approche multivue pour la conception des SI à base de composants. *Proceeding of* INFORSID'O5, Grenoble France.
Lankhorst, M. (2005). *Enterprise architecture at work: Modelling, communication and analysis*. Springer
Leite, J.C.S.P. Freeman, P.A. (1991). Requirements validation through viewpoint resolution. In *IEEE Transaction in Software Engineering*. 17(12),1253-1269
Lissandre, M. (1990). *Maîtriser SADT*. Armand Colin
Lopes, C.V. Kiczales, G. (1998). Recent Developments in AspectJ. in *Proceedings of ECOOP'98 Workshop Reader*. Springer-Verlag LNCS 1543
Malenfant, J. (1996). Abstraction et encapsulation en programmation par prototypes. in *Technique et Science Informatiques*. 15, 6, 709-733. Hermes
Marca, D.A. Mc Gowan, C.L (1987). *SADT, Structured Analysis and Design Technique*. Mc Graw Hill
Mariño, O. (1993). *Raisonnement classificatoire dans une représentation à objets multi-points de vue*. Thèse de l'université Joseph Fourier, Grenoble 1, Octobre 1993.
Masini, G. Napoli, A. Colnet, D. Léonard D., Tombre, K. (1989). *Les langages à objets*. InterEditions, Paris
McDirmid, S. Hsieh, W. (2003). Aspect-Oriented Programming with Jiazzi. In *Proceedings of Int'l Conference Aspect-Oriented Software Development*, 70-79
Mili, H. Mcheick, H. Sadou, S. (2002). CorbaViews-Distributing Objects that Support Several Functional Aspects. *in Journal of Object Technology, Special Issue:TOOLSUSA Proceedings.* 1(3), 207-229
Minsky, M. (1975). A Framework for Representing Knowledge. in *The Psychology of Computer Vision* New York:6,156-189. P.H. Winston, McGrawHill
Motro, A. (1987). Superviews:Virtual integration of multiple databases. In *IEEE Transactions On Software Engineering*. SE-13(7):785–798
Mullery, G. (1979). CORE - A method for controlled requirements specification. In *Proceedings of the Fourth International Conference on Software Engineering*. Munich, RFA:126–135. IEEE Computer Society Press
Mullery, G. (1985). Acquisition – Environment. In *Distributed Systems: Methods and Tools for Specification*. LNCS, 190, Springer-Verlag. M. Paul and H. Siegert
Nassar, M. (2005). *Analyse/conception par points de vue : le profil VUML*. Thèse de l'Institut National Polytechnique de Toulouse
Nguyen, G.T. Rieu, D. (1992). Multiple Object Representations. *Proceedings of 20th A.C.M Computer Science Conference*. Kansas City
Nuseibeh, B. Kramer, J. Finkelstein, A.C.W. (1994). A framework for expressing the relationships between multiple views in requirements specification. in *IEEE Transactions on Software Engineering*. 20(10),760 –773
Nuseibeh, B. (1994). *A Multi-Perspective Framework for method integration*. Thèse de doctorat de l'Université de Londres
O2Views (1995). *O2Views User Manual*. Projet VERSO, INRIA Rocquencourt. version 2. Le Chesnay, France
Ohori, A. Tajima, K. (1994). A polymorphic Calculus for Views and Object Sharing. In *Proceedings of the 13th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. 255-266
Ossher, H. Kaplan, M. Harrison, W. Katz, A. Kruskal, V. (1995). Subject-Oriented Composition Rules, in *Proceedings of OOPSLA'95, ACM Sigplan*. USA, Austin. 235-250
Ossher, H. Tarr, P. (2000). Hyper/J : Multi-Dimentionnal Separation of Concern for Java, in *Proceedings of ICSE'00*. Limerick, Ireland. ACM Press, C. Ghezzy
Putman, J-R. (2000). *Architecting with RM-ODP*. Prentice Hall

Rich, C. (1981). Multiple Points of View in Modelling Programs. In *SIGPLAN Notices*. 16(1),177-179, SIGPLAN & ACM Press

Richardson, J. Schwarz, P. (1991). Aspects : Extending Objects to Support Multiple, Independent Roles. In *Proceedings of the ACM SIGMOD Int'l Conference On Management of Data*. 298-307

Rundensteiner Elke, A. (1992). MultiView A Methodology for Supporting Multiple Views in Object-Oriented Databases. In *Proceedings of the 18th International Conference on Very Large Data Bases (VLDB'92)*. Vancouver, Canada

Scholl, M.H. Laasch, C. Rich, C. Schek, H.-J. Tresch, M. (1991). *The COCOON Object Model*. Technical Report 211, Department of Computer Science, Suisse, Zurick

Shaw, M. Garlen, D. (1996). *Software Architecture : Perspectives on an Emerging Discipline*. Prentice Hall

Sheth, A.P. Larson, J.A. Watkins, E. (1988). TAILOR, a tool for updating views. In *Proceedings of Int'l Conf. on Advances in Database Technology (EDBT)*. Venise, Italie:190–213,

Shilling, J.J. Sweeney, P.F. (1989). Three Steps to Views: Extending the Object-Oriented Paradigm. actes de de la conférence OOPSLA'89. 353-361

Smith, G. (2000). The object-Z specification language. In *Advances in formal methods*. Kluwer Academic Publishers

Sommerville, I. Sawyer, P. (1997). *Requirements engineering: A good practice guide.* Wiley

Souza dos Santos, C. Abiteboul, S. Delobel, C. (1994). Virtual Schemas and Bases. *In Advances in Database Technology EDBT'94*. 779, 81-94, Cambridge, Royaume-uni. Springer-Verlag. Jarke, M., Bubenko Jr., J. A., et Jeffery, K. G.

Souza dos Santos, C. (1995). Design and Implementation of Object-Oriented Views. in *Proceedings of DEXA'95*. 978, 91-102. Londres, Royaume-uni. Springer

Stefik, M. Bobrow, D.G. (1985). Object-Oriented programming : Themes and variations. in *A.I. magazine*, 6(4),40-62

Vanwormhoudt. G. (1999). *CROME : un cadre de programmation par objets structurés en contextes.* Thèse de Doctorat, Lille, France, Septembre 1999.

Wieringa, R.J. Blanken, H.M. Fokkinga, M.M. Grefen, P.W.P. (2003). Aligning application architecture to the business context. In *Proceedings of CaiSE*. 2681, 209-225

Wortmann, J. (2001). *Concurrent Requirements Engineering with a UML Subset based on Component Schema Relationships*. Dissertation, FB Informatik, Berlin