REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE MENTOURI DE CONSTANTINE
FACULTE DES SCIENCES DE L'INGENIEUR
DEPARTEMENT D'INFORMATIQUE

N° d'ordre :
Série :

THESE

Présentée pour obtenir le diplôme de Doctorat d'Etat

Intégration des points de vue dans les bases de données à objets : Le modèle Multi-Viewpoint-DataBase

---

Présentée par :

Mme Fouzia BENCHIKHA Epouse MAGRA

Dirigée par :

Pr. Mahmoud BOUFAIDA

SOUTENUE LE / /2007

Devant le jury

Président :	Zaidi SAHNOUN	Professeur à l'Université Mentouri de Constantine
Rapporteur :	Mahmoud BOUFAIDA	Professeur à l'Université Mentouri de Constantine
Examinateur :	Mohamed LASKRI	Professeur à l'Université de Annaba
Examinateur :	Noureddine DJEDI	Professeur à l'Université de BISKRA
Examinateur :	Zizette BOUFAIDA	Professeur à l'Université Mentouri de Constantine

Remerciements

Au terme de ce travail, je tiens à exprimer ma profonde reconnaissance au Pr M. Boufaida qui a été mon encadrant pendant toutes ces années de thèse. Je le remercie pour sa dévotion, ses conseils et son œil critique qui m'ont orienté pour mener à bien mes travaux de recherche.

Mes remerciements vont au Pr Z. Sahnoun qui a accepté de présider le jury et à tous les membres de jury qui ont accepté d'être des examinateurs. Je suis honoré à l'attention et au temps qu'ils vont consacrer à ce travail.

Sommaire

Chapitre 2 : Les points de vue dans les bases de données à objets

Chapitre 3 : MVDB : Un modèle basé point de vue pour une représentation multiple, évolutive et distribuée des objets

Chapitre 4 : Caractérisation des mécanismes de partage, de protection et de cohérence des données dans le modèle MVDB

Chapitre 5 : Mise en œuvre et extension de l'approche MVDB

Introduction générale

1. Contexte général

Un des objectifs principaux des recherches en bases de données (BD) est de fournir des modèles de données les plus expressifs et les plus extensibles possibles. Expressifs de manière à restituer la réalité modélisée en tentant d'être le plus fidèle à celle-ci et en tenant compte des contraintes d'efficacité inhérentes à toute application informatique. Extensibles en ce sens que le concepteur doit pouvoir ajouter ses propres éléments liés au contexte d'utilisation.

Le modèle "objet" a été proposé dans cette ligne de pensée. On reconnaît pour ce dernier la puissance d'expression, de structuration et de flexibilité qu'en à la modélisation des objets du monde réel. Cependant, au-delà des concepts importants offerts par le modèle objet, certains problèmes restent ouverts, notamment tous ceux liés à la représentation multiple des entités et à l'évolution de leur structure[1] au sein d'un système informatique. On parle très souvent du concept de perspective, aspect, dimension, vue ou encore point de vue.

Les systèmes d'information présents dans divers domaines d'application comme l'informatique de gestion, le génie civil, l'architecture, l'information géographique, la biologie, la CAO, etc. fournissent plusieurs exemples dans lesquels les thématiciens du domaine décrivent les problèmes rencontrés pour représenter de façon multiple les entités modélisées.

Ainsi, parmi les exemples dont la notion de point de vue caractérise un aspect des entités du domaine, nous citons :

- en génie civil, un pont peut être un projet communal pour les autorités, une figure géométrique pour le géomètre, une maquette pour l'architecte, et une entité physique pour l'entrepreneur, tout en conservant des informations communes (lieu, nom, etc.),
- en informatique de gestion, un acte d'achat est un devis pour le vendeur, une commande pour le service commandes et une facture pour le service comptabilité,
- en informatique géographique, les objets peuvent exhiber des représentations différentes selon l'échelle d'une carte, son type, sa finalité, etc.
- en électronique, une puce peut être vue comme une spécification pour le vendeur, un diagramme logique pour le bureau d'études ou bien un pavé de silicone pour le service fabrication. Ainsi, une même entité peut prendre différentes formes et être considérée sous différentes perspectives selon le point de vue de l'observateur.

L'importance de gérer les points de vue a suscité l'intérêt de plusieurs communautés scientifiques. C'est ainsi que de nombreux travaux ont été menés, parfois parallèlement, dans des directions identiques. On peut donc dire que l'étude des points de vue est une notion horizontale, c'est-à-dire qu'elle n'est pas propre à un domaine, mais est plutôt considérée comme une notion intégrable aux modèles de données quelle qu'en soit leur

[1] Nous ne considérons pas ici l'évolution du comportement des entités, ceci relève d'un autre aspect traitant le cycle de vie de ces dernières.

origine. La genèse de l'étude des points de vue n'est pas précise, et il est très difficile, voire impossible, d'attribuer la primauté des travaux à certains plutôt qu'à d'autres.

D'une manière générale, la notion de points de vue s'intéresse à la variété de perceptions portées sur un même univers de discours. Elle s'oppose à l'approche monolithique et permet de modéliser une même réalité selon des points de vue différents. Un point de vue dans son sens large, étant la perception qu'une personne en fait d'un monde observé.

2. Problématique

L'origine de notre réflexion menée rejoint de nombreuses préoccupations liées au problème de la représentation multiple des données. Concevoir le fait qu'un objet puisse acquérir plusieurs descriptions structurelles qui se complètent est une démarche naturelle et les points de vue constituent potentiellement de bons candidats pour prendre en compte cet aspect.

Le concept de point de vue constitue un sujet actif de recherche comme en témoignent les différents travaux et les nombreuses publications tant en représentation des connaissances qu'en génie logiciel, en modélisation des systèmes complexes, en programmation, en bases de données, etc.

Dans le domaine des bases de données, l'ouverture vers les nouvelles applications de complexité accrue d'une part et l'expansion des systèmes distribués d'une autre part nécessite de revoir la manière de concevoir le schéma d'une base de données en prenant en compte sa description selon différentes perspectives. En effet, au vue des exemples cités ci-dessus, un nouveau besoin de structuration des données surgit. Ce besoin de considérer les points de vue dans un système d'information apparaît clairement dans les techniques des vues des bases de données. La nécessité d'élaborer et d'exploiter des schémas externes selon les utilisateurs est assez ancienne : l'architecture ANSI/X3/SPARC remonte quasiment aux débuts des travaux sur les bases de données relationnelles. Elle concerne essentiellement la phase d'exploitation des bases de données. Les méthodes de conception par intégration des vues ont fait leur apparition à la même époque (1978). Rappelons qu'elles ont pour objectif de construire le schéma conceptuel à partir d'un ensemble de schémas-vues d'utilisateurs.

Cependant, l'utilisation des mécanismes de vues n'est pas une solution au problème de la représentation multiple à laquelle nous nous intéressons. En effet, une vue est un schéma dérivé qui permet seulement de filtrer, de masquer et de calculer des données spécifiques à partir de données existantes. De ce fait, et comme le fait remarquer [37], un schéma, même pourvu de vues, est une vision très centralisée de l'univers de discours; ce qui ne favorise pas une relative liberté de description des différentes vues.

Dans la même perspective qui consiste à doter les objets de plusieurs descriptions structurelles et comportementales, nous enregistrons les travaux sur l'intégration du mécanisme de rôle dans les modèles de bases de données. Ce mécanisme permet aux objets d'avoir plusieurs rôles en même temps et peut changer sa structure et son comportement pendant son cycle de vie. La première tentative d'intégration des rôles dans le modèle réseau remonte à 1977et est attribuée à Bachman et Daya [8]. Plus tard, avec l'avènement de l'objet, et consécutivement aux problèmes posés par la conservation des modèles de SGBD objets standard, plusieurs travaux proposent d'intégrer la gestion des rôles au niveau des modèles eux-mêmes. Ainsi, en 1987, le premier système autorisant une instance à être rattachée à plus d'une classe est IRIS

[41]. Dès lors de nombreuses propositions se sont succédées. Cependant, le but des rôles est de fournir la possibilité aux objets d'évoluer sans affecter le schéma de la base de données, ce qui ne répond pas au besoin de structuration d'un schéma de base de données selon des points de vue différents qui sera systématisé de la conception du schéma jusqu'à son exploitation.

3. Objectif de la thèse

L'objectif de notre thèse est d'apporter une réponse satisfaisante à ce besoin de structuration en proposant le modèle objet MVDB (MultiView DtaBase). Ce dernier permet d'intégrer et de gérer le concept de point de vue de façon :

- ***inhérente***, en considérant le point de vue comme un concept de base qui représente une partie intégrante du modèle et non comme un mécanisme greffé sur celui-ci. Cependant, l'utilisateur concepteur d'application ne devant pas être submergé par les artefacts mais posséder des outils adéquats et intégrés,
- ***compatible***, avec le modèle standard des bases de données objets,
- ***intuitive,*** dans le sens où les points de vue doivent être facilement modélisables et exploitables,
- ***distribuée***, en offrant une approche décentralisée de conception d'un schéma de base de données objet. Ainsi ce dernier est considéré comme un ensemble de représentations partielles, complémentaires et interdépendantes. Chaque représentation considère un aspect de la description des entités du domaine selon un point de vue donné,
- ***sûre***, en offrant les mécanismes adéquats pour garantir une conception intègre du schéma et assurer la cohérence globale des données.

Au-delà de la définition d'un modèle de bases de données, nous tentons de :

- en guise de validation, réaliser une mise en œuvre en langage java pour une conception et une gestion aisée de bases de données multi-points de vue,
- en guise d'extension de notre travail, proposer une architecture support et une méthode d'intégration de nouveau composant. Nous y reviendrons plus en détails dans nos contributions.

4. Pourquoi est-ce important ?

Les besoins des utilisateurs, et par conséquent les applications, sont rarement figés, et de nouvelles fonctionnalités demandent sans cesse à être intégrées dans un système qui veut refléter la réalité. C'est une caractéristique qui est malheureusement très difficile à prendre en compte, car on est souvent obligé d'effectuer un choix entre d'un côté la sécurité, la rapidité, et la cohérence des applications, qui est une caractéristique nécessaire voire indispensable dans les systèmes d'informations à l'heure actuelle, et d'un autre côté la souplesse en termes de prise en compte de l'évolution structurelle des données, qui est une caractéristique dont l'importance grandit, engendrée par des impératifs tant scientifiques que techniques.

5. Contributions

Notre thèse s'inscrit dans le cadre des travaux sur les modèles de bases de données à objets. Le modèle que nous proposons est une extension du modèle objet à base de classes avec le concept de point de vue et qui est compatible avec le standard de l'ODMG. Il permet de considérer le schéma d'une base de données comme une spécification qui tient compte de plusieurs points de vues. Chaque point de vue représente un aspect de la description des données et est détenu par une base de données indépendante dite "partielle". Il s'agit donc d'un développement fondé sur l'élaboration décentralisée (répartie) de bases de données représentant un même univers de discours. En tenant compte des points abordés ci-dessus notre modèle MVDB (Multi-View DataBase) se base sur les principes suivants :

- le schéma d'une base de données est composé d'un ensemble de schémas point de vue (schéma-PV). Chaque schéma-PV décrit un aspect de la description des données et est détenu par un système de base de données indépendant.
- la construction des schémas-PV est basée sur un schéma de base appelé le référentiel. Ce dernier détient les propriétés de base des entités de l'univers de discours et qui sont partagées par tous les schémas-PV. Chaque schéma-PV peut décrire tout l'ensemble ou un sous ensemble des entités du référentiel selon un point de vue donné.
- les objets qui sont les instances dans les différents points de vue contiennent les données du domaine d'étude (ou du système). Un objet peut être représentatifs dans plusieurs points de vue. Le lien entre les différentes instances d'un objet est réalisé à travers la notion d'identité d'objet. Un objet est identifié d'une manière unique dans tous les schémas-PV. L'univers complet d'un système est l'union des différents schemas-PV.
- la représentation multiple est décentralisée afin de permettre une libérté de spécification des différentes représentations.
- les représentations partielles s'échangent les informations entre elles, pour cela nous proposons de définir explicitement des relations qui permettent à une représentation d'accéder à certaines informations spécifiées dans une autre.
- la représentation multiple est cohérente. Une étude de la cohérence locale au niveau de chaque schéma des représentations partielles ainsi que l'étude de la cohérence globale du schéma multi-points de vue sont menées.

Afin de valider les concepts de notre modèle, une mise en œuvre est réalisée en langage java et appliquer sur une étude de cas simple. Des outils de conception distribuée d'un schéma de base de données multi-points de vue sont mis à la disposition des concepteurs de la base. Ces outils sont disponibles via des browsers qui permettent d'une manière souple, flexible et sûre de générer automatiquement des schémas de bases de données multi-points de vue et de les peupler avec des objets ayant plusieurs structures selon leur représentation dans les différentes bases de données partielles.

6. Plan de la thèse

Le mémoire est organisé en cinq chapitres :

- Dans le premier chapitre, nous introduisons des définitions générales du concept "point de vue". Nous y dressons une étude bibliographique sur des travaux en représentation des connaissances, en génie logiciel, à la programmation et à la modélisation des systèmes complexes. Nous présentons les différents mécanismes et techniques qui ont été adoptés dans les différents domaines pour prendre en compte cette notion.
- Le deuxième chapitre est consacré à l'étude du concept de point de vue dans le cadre des bases de données et plus particulièrement celles basées sur le modèle objet. Nous examinons les techniques de vues et celles de rôles qui tentent de modéliser les points de vue. Nous dégageons les apports et les limites de ces approches, et déterminons quels éléments de réponse ils apportent à notre problème. Une étude comparative des trois mécanismes de vue, rôle et point de vue est faite à la fin du chapitre afin de positionner notre approche par rapport à l'existant.
- L'approche MVDB fait l'objet du troisième chapitre. Nous y expliquons et formalisons les différents concepts de l'approche qui ont été présentés ci-dessus. Nous illustrons nos propos au travers de l'exemple d'un système d'information d'une entreprise de fabrication et de distribution de produits.
- Dans le quatrième chapitre, nous nous intéressons aux mécanismes de partage et de cohérence des données qui sont inhérents à notre modèle. Dans un premier temps, nous présentons et définissons formellement le concept de "passerelle" qui permet l'échange des données entre points de vue. Le partage des données est également réalisé au niveau des objets en adaptant le mécanisme de la délégation que nous empruntons des modèles à prototypes. La deuxième partie du chapitre est consacrée à la cohérence des données. Nous proposons une typologie des contraintes pour le modèle MVDB, ainsi qu'un langage d'expression de ces contraintes.
- Le dernier chapitre est consacré à la mise en œuvre de l'approche MVDB en utilisant le langage java. Le prototype java-MVDB est réalisé et permet de générer des schémas de bases de données avec de multiples points de vue. Le schéma est conçu d'une façon distribuée. Nous utilisons le langage XML pour tout échange de documents (données) entre les différents sites. Enfin, nous présentons une architecture qui permet le support de la gestion des bases de données multi-points de vue basée sur les bases de données fédérées.

Chapitre 1

Où l'on présentera l'approche par point de vue dont la naissance a ouvert de nouveaux horizons sur la manière de modéliser le monde réel avec toute la complexité, l'irrégularité et la richesse en informations qu'il représente.

En effet, pour produire, il faut d'abord modéliser[1]*. Modéliser consiste à faire une abstraction d'une réalité riche en information. Cette abstraction dépend de l'agent qui la réalise, de son domaine d'intérêt et de ses connaissances préalables.*

Sowa [84] rapporte un exemple de modélisation multiple décrit par Kierkegaard[2]*. Il s'agit de la description d'une forêt selon les approches de différentes personnes. L'approche écologique verra la forêt comme un système complexe de flore et de faune interdépendantes. L'approche esthétique mettra en avant la diversité des odeurs, des sons et des vues. L'approche industrielle y voit une source de matière pour la fabrication de papier, alors qu'une vue mystique de la forêt la considère comme une manifestation de l'harmonie de l'univers. Aucune de ces vues ne détient la vérité absolue, chacune exprimant un aspect de la réalité. Pour Kierkegaard, chaque aspect exclut les autres. En poussant la relativité des conceptualisations jusqu'au bout, il en arrive à la constatation plutôt pessimiste que chaque vue peut avoir une cohérence interne, mais que les vues (et les systèmes conceptuels) de deux individus peuvent être incomparables.*

Entre les deux positions extrêmes de la vue unique d'un coté et de l'incompatibilité totale des vues de l'autre, il semble y avoir suffisamment de place pour l'élaboration constructive de ***modélisation multiple****. Plutôt que d'avoir des vues qui s'excluent, on cherchera à les mettre en rapport, afin de disposer de plusieurs façons de comprendre un domaine, ou encore de résoudre certains problèmes qui s'y posent d'où la genèse du concept de* ***point de vue****.*

L'approche par points de vue ou multipoints de vue a été introduite et utilisée dans plusieurs domaines de l'informatique aussi variés que les bases de données, la représentation des connaissances, l'analyse et la conception, les ateliers et les environnements de génie logiciel et les langages à objets.

Dans ce chapitre, nous nous intéressons plus particulièrement à ce concept qui apparaît, avec certaines variations, sous plusieurs termes et aspects selon son abord dans les différents domaines. Nous en donnons quelques définitions dans quelques domaines où il a été une partie intégrante d'un modèle, d'une approche, d'un langage ou encore d'un système. Puis nous passons en revue quelques travaux significatifs issus des domaines de la représentation des connaissances, du génie logiciels, de la programmation et de la modélisation des systèmes complexes.

---

[1] Nous employons le terme "modélisation" dans son sens le plus général comme l'ingéniosité de savoir ou de faire sur le monde réel.

[2] Kierkegaard, Soren (1843) Either/Or, two vols., Anchor Books, Garden City, NY.

1. Le concept de point de vue

En informatique, la plupart des systèmes de modélisation des données (en représentation des connaissances, en programmation, en modélisation des systèmes complexes, en bases de données, etc.) négligent la variété de perceptions portées sur un même univers de discours et offrent des outils pour créer un modèle unique pour une vision unique du monde observé, une représentation étroite où chaque utilisateur doit filtrer ce qui l'intéresse. L'approche par point de vue s'oppose à cette approche mono-point de vue, ou monolithique et permet de modéliser une même réalité selon des points de vue différents. Un point de vue dans son sens large, étant la perception qu'une personne en fait d'un monde observé.

Actuellement, on s'accorde à reconnaître l'intérêt de la notion de point de vue dans la conception et le développement des applications multi-utilisateurs qui requièrent la coopération de plusieurs experts voire des utilisateurs, avec chacun ses centres d'intérêts et ses connaissances. Cependant, cette notion reste encore vague et très variée dans sa définition et dans son utilisation comme nous le verrons ci-dessous.

1.1. Qu'est-ce qu'un point de vue ?

Le concept de point de vue revêt des significations différentes selon son abord dans les différents domaines de l'informatique. Il y apparaît, avec certaines variations, sous des termes tels que **perspectives**, **rôles**, **aspects**, **vues**, **dimensions**, **contextes** et **points de vue**. Dans la suite nous en donnons quelques définitions recueillies de travaux de recherche sur l'intégration des points de vue dans certains domaines.

Dans le domaine de la représentation des connaissances, O. Marino dans [56] écrit que "*la perspective d'un monde est la perception qu'en a un expert d'une discipline particulière*". Et pour G.T. Nguyen et D. Rieu [68], "*le terme de point de vue est synonyme de celui de représentation d'un objet qui possède plusieurs représentations et est soumis à plusieurs utilisateurs*".

Dans le domaine du génie logiciel, M. Stefik et D.G. Bobrow [87] définissent le terme de point de vue comme "*une forme d'objet composite interprété comme différentes vues sur la même entité conceptuelle*". Plus tard, D.A. Marca et C.L. McGowan dans le définissent comme "*la perspective selon laquelle un problème est étudié*".

Dans le domaine des bases de données, S. Heiler et S.B. Zdonik [47] écrivent qu'"*une vue est un contexte dans lequel des fonctions sont appliquées à des objets*". Et pour J.J. Shilling et F.P. Sweeney [82] "*un point de vue peut être défini comme une abstraction simplifiée d'une structure complexe en supprimant l'information non pertinente pour l'objectif courant*".

Enfin dans le domaine de la modélisation et des systèmes distribués, X. Cueigner et V. Lextrait [31] donnent la définition suivante : "*une vue dans VSL est définie comme un modèle cible associé à des spécifications sémantiques sur un modèle originel qui spécifie comment le premier est déduit du second*". Et selon B. Neseibeh, J. Kramer et A. Finkelstein [6] définissent "*les points de vue sont des objets distribués, localement gérés et faiblement couplés encapsulant des connaissances sur les modes de*

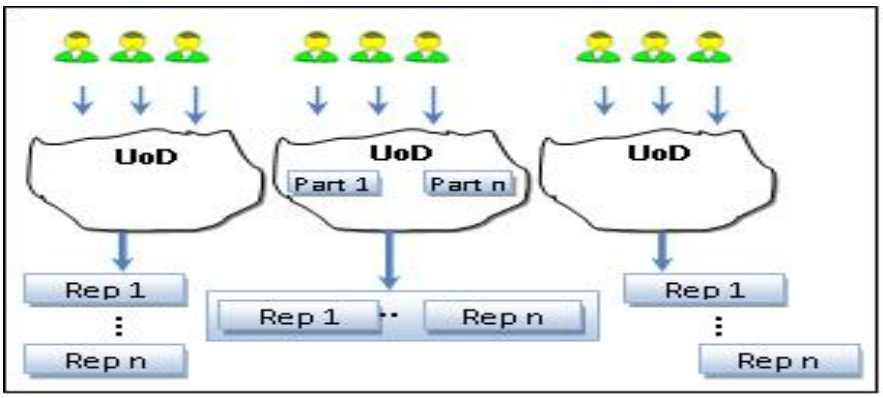
représentation, sur le processus de développement et sur les spécifications partielles d'un système et de son domaine".

En toute généralité, l'approche par point de vue est fondée sur la conjonction acteur/information. Donc, il faut inclure l'acteur dans l'action. Nous définissons donc un point de vue comme "*une position conceptuelle mettant en liaison d'une part un acteur qui observe et d'autre part un phénomène (ou un monde) qui est observé*". Les acteurs peuvent observer un même univers de discours produisant des points de vue qui peuvent être considérés de différentes manières (voir illustration en figure 1) :

1. **Points de vue uniformes** : ce cas correspond à la représentation mono-disciplinaire, c'est-à-dire tous les acteurs ont la même vision de l'univers de discours et produisent des représentations équivalentes. Par exemple, considérons différentes équipes de conception, chacune utilise un modèle de données différent qu'elle juge le meilleur pour la réalisation d'un projet.

2. **Points de vue complémentaires** : dans ce cas chaque acteur qui observe le même monde observé par les autres acteurs en voit une partie. Chaque point de vue est une représentation partielle et cohérente du monde. Les différentes représentations qui découlent des différents acteurs sont alors complémentaires, leur union est une représentation complète et cohérente du monde.

3. **Points de vue comparables** : dans ce cas les acteurs produisent des représentations comparables au sens plus général/spécifique. Chaque agent a une réalité qui inclut sa connaissance du monde qui l'entoure et en donne une perception selon sa vision.



a) Points de vue uniformes b)Points de vue complémentaires c)Points de vue comparables

Figure 1. Les différentes natures des points de vue sur un même univers de

Dans le cadre de notre étude nous nous intéressons à la deuxième interprétation de la relation "acteur-monde", qui suppose que les différents points de vue sur un même univers de discours sont des visions *partielles* mais *complémentaires.* Leur union produit une représentation complète et cohérente du monde.

1.2. Pourquoi les poins de vue ?

La notion de point de vue a été utilisée à des fins multiples dans les différents domaines de l'informatique. L'intérêt apporté au mécanisme de point de vue croit de jour en jour. Malheureusement, aucun consensus n'est encore atteint en ce qui concerne son intégration dans les différents domaines. Le problème réside dans le fait que cette notion est utilisée dans plusieurs contextes et à des fins multiples et variées.

En littérature, les travaux traitant la notion de point de vue en modélisation orientée objet sont plus pragmatiques. Dans ce qui suit nous résumons quelques objectifs orthogonaux pour lesquels les points de vue ont été exploités en informatique.

1) Le point de vue comme un moyen de description multiple des entités :

Le concept de point de vue semble résulter naturellement de la multiplicité des regards portés sur les objets d'un domaine d'étude. En effet, une entité du monde réel peut avoir plusieurs contextes comportementaux et plusieurs états d'où la notion de « description multiple ». Cette dernière confère à un même univers de discours plusieurs descriptions partielles. Chaque description fournit un point de vue. Les descriptions partielles sont complémentaires et produisent une description complète et cohérence des entités du monde réel. Une des premières références à la notion de description multiple se trouve dans le travail de Minsky [63], dont l'intérêt principal était le raisonnement dans un système de perception et de vision par ordinateur. Récemment, le paradigme des points de vue a été appliqué dans le web pour la représentation et l'affichage multidimensionnel des données, les informations supposent avoir plusieurs facettes sous différents contextes [85][86].

2) Le point de vue comme un moyen pour maîtriser la complexité des systèmes:

De nombreux travaux de recherche se basent sur le concept de point de vue avec pour objectif principal de prendre en compte explicitement la complexité de certains systèmes. L'analyse d'un domaine est alors menée par découpage en descriptions partielles selon des aspects différents et parfois complémentaires. Ainsi, dans le contexte d'un environnement de développement de logiciels Schilling et Sweeney [82] suggèrent des vues multiples comme des abstractions destinées à simplifier la structure complexe d'un système. Chaque vue fournit une interface adaptée à un utilisateur (et/ou développeur) particulier du système. Dans le domaine de l'apprentissage de la programmation, on trouve le système EXPLAINER [73] qui décrit des programmes informatiques selon différents aspects (texte source, représentation graphique, exemples d'utilisation, etc.).

3) Le point de vue comme une approche de modélisation et de développement décentralisé des systèmes :

Plusieurs auteurs estiment que la modélisation des systèmes surtout complexes tels que définis dans [54] ne peut s'appréhender avec les mêmes techniques que pour les petits systèmes dont le développement est maîtrisable par un nombre restreint de personnes. Différents travaux [51][32] proposent une approche répartie de développement fondée sur les points de vue. En effet, la modélisation d'un système complexe ne peut être une tâche centralisée basée sur un formalisme unique. Il faut

alors laisser la possibilité aux différents concepteurs d'un système de développer de façon décentralisée. Tout processus de développement peut donc être représenté par un ensemble de points de vue corrélés. Des solutions généralement fondées sur les systèmes logiques sont utilisées pour établir cette corrélation. L'approche VBOOM (View Based Object Oriented Methodology) [51] est l'exemple type de la méthode d'analyse/conception intégrant le mécanisme de point de vue en introduisant le concept de modèle visuel. La nécessité d'introduire les points de vue lors de la modélisation s'exprime également dans des méthodes telles que SADT [76].

4) Le point de vue comme un mécanisme efficace pour la technologie orientée objet :

Il est communément reconnu que la technologie orientée objet a apporté un net progrès dans la modélisation des systèmes complexes par son pouvoir d'expression et sa meilleure réutilisabilité. Cependant, elle s'est vite accompagnée de l'émergence de nouveaux besoins tels que la nécessité de distinguer les données en provenance de différents concepteurs ou encore de l'évolution des objets ainsi que leur (re)classification multiple et dynamique. La rigidité d'un objet tant en ce qui concerne son état que son comportement a été très vite remise en cause via les perspectives de KRL [18], les contextes dans CROME [32] et les points de vue dans TROPES [56].

5) Le point de vue comme un mécanisme de résolution de problèmes :

Le concept de point de vue apporte des solutions simples et satisfaisantes à des problèmes épineux posés dans les différents domaines de l'informatique. En représentation des connaissances, par exemple, le point de vue est introduit dans la classification multiple des objets [12] [34] [75], dans la recherche de valeurs héritées [72], dans la modélisation de concepts indépendants et dans le traitement des conflits d'héritage multiple [34]. En modélisation, la prise en compte explicite des points de vue des différents concepteurs pour produire un modèle unique partageable est un moyen efficace pour améliorer la cohérence de la modélisation [24].

Dans le contexte de notre travail, les points de vue sont utilisés essentiellement comme un mécanisme pour une représentation multiple des objets, comme un mécanisme pour le traitement des contraintes d'intégrité et comme une approche de conception distribuée de schéma de base de données à objets.

Dans les sections suivantes, nous abordons quelques travaux significatifs intégrant le concept de point de vue dans le domaine de la représentation des connaissances, du génie logiciel, de la programmation et de la modélisation des systèmes complexes. En fait, c'est en représentation des connaissances que la notion de point de vue a tout d'abord été considérée. Nous présentons ci-dessous quelques propositions et des mécanismes permettant de la prendre en compte.

2. Prise en compte des points de vue : premières propositions en représentation des connaissances

La représentation des connaissances était le domaine précurseur dans l'abord et l'introduction de la notion de point de vue dans des modèles à objets en raison du

caractère multi-disciplinaire des applications. En effet, la plupart des applications en intelligence artificielle nécessitent l'intervention de plusieurs experts ayant chacun une connaissance particulière du domaine d'étude qui procure aux objets manipulés une représentation et un type de raisonnement particuliers. Les systèmes d'aide à la décision ou encore les applications de conception assistée par ordinateur sont typiquement des applications multi-disciplinaires.

Chacune des premières propositions en représentation des connaissances a mis l'accent sur une technique ou un principe pour gérer la représentation multiple qui a par la suite été repris dans de nombreux modèles de points de vue. C'est essentiellement dans ce but que nous nous intéressons à ces premières propositions. Au delà de la présentation de trois approches qui exemplifient assez bien les différentes façons de représenter explicitement les points de vue, nous discutons dans cette section le rapport qui existe entre le concept de point de vue et des notions similaires plus répandues dans le monde informatique du développement et de la modélisation telles que l'héritage multiple, la composition et la délégation.

2.1. Points de vue et spécialisation

De prime abord, il parait naturel d'examiner l'adéquation de l'héritage (spécialisation) à l'implantation des points de vue, étant donnée leur proximité sémantique. Cependant, les différentes études ont démontré son inadéquation à une modélisation dynamique et flexible des points de vue multiples d'un objet. Dans ce qui suit, nous présentons d'abord l'approche du langage KRL pour la représentation multiple des objets, puis nous examinons son rapport avec le mécanisme d'héritage et les limites de celui-ci pour l'implantation des points de vue.

2.1.1. Les perspectives dans le langage KRL

KRL (Knowledge Representataion Language) [18] est un modèle mixte de représentation des connaissances qui permet d'exprimer la connaissance sous forme procédurale et déclarative. Il est un des premiers langages de représentation des connaissances avec des frames à reconnaître qu'un objet peut être vu de plusieurs façons, selon le point de vue de l'observateur. KRL possède un niveau méta, qui décrit l'ensemble des concepts du modèle par des méta-classes appelées des *catégories*. Le terme *unité* est utilisé pour désigner les entités à différents niveaux d'abstraction (classe, objet, relation, événement, …). Il y'a sept types différents de catégories d'unités : *Basic*, *Specialization*, *Individual*, *Abstract*, *Manifestation*, *Relation* et *Proposition*. Les quatre premières servent à définir des *perspectives*. La catégorie *Basic* permet de partitionner l'univers du discours en des familles disjointes d'objets (tels Personne, Véhicule). Un objet particulier ne peut appartenir à deux unités *Basic*. *Abstract* est la catégorie des unités abstraites, qui factorisent des propriétés à hériter par leurs spécialisations. Les unités de la catégorie *Specialization* spécialisent les catégories *Basic* ou les catégories *Specilalization*. La catégorie *Individual* décrit des entités réelles : des individus.

Une *unité* de KRL est un ensemble de descripteurs. Pour traiter les perspectives, KRL utilise un descripteur particulier appelé *perspective*. Ce dernier permet de modéliser un point de vue sur un objet. Le descripteur *perspective* est défini par son prototype qui est une classe de type *Basic*, *Abstract* ou *Spécilaization* et par une

évaluation des attributs du prototype. Les perspectives ou les points de vue sont donc représentés au niveau des instances. 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.

Pour la clarté de l'exposé, nous utilisons dans cette section le même exemple pour illustrer les différentes approches. Cet exemple consiste à modéliser les employés d'une faculté qui peuvent avoir différentes activités. On s'intéresse, en particulier, aux activités de l'enseignement et de la recherche.

La figure 2 présente, dans le formalisme KRL, l'unité Employé qui est une unité de base de type *Basic* spécialisée en plusieurs unités de type *Specialization* telles que Maître-conférence, Professeur, Assistant, Art, Sciences Pures, Sciences Sociales, etc. l'unité individuelle, Benali est définie par trois descripteurs de type *perspective* qui sont : Employé, Assistant et Art. Chaque descripteur est une caractérisation indépendante de l'objet associé. Ainsi, pour parler de Benali on peut dire : "un employé de l'université", "un assistant", "un enseignant de musique", etc.

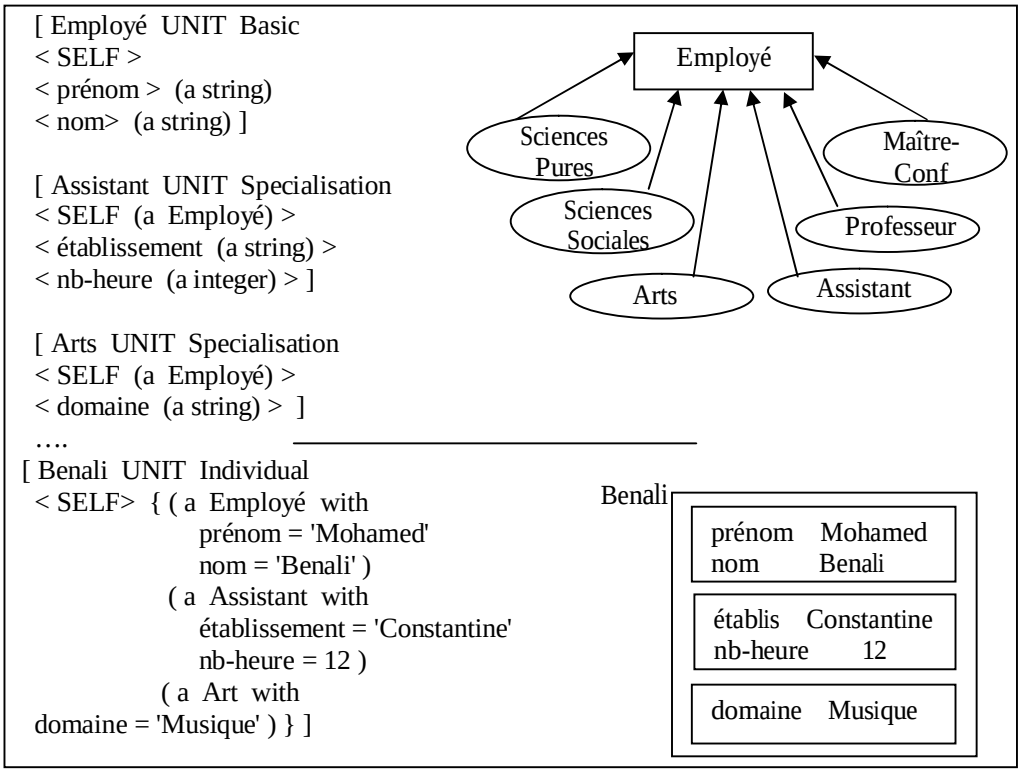


Figure 2. La représentation multiple dans le langage KRL

2.1.2. Spécificités de la représentation multiple dans KRL

KRL permet la prise en compte des points de vue exprimés par le terme "perspectives". Les perspectives apparaissent au niveau des entités réelles ou des individus et se caractérisent par :

- 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 celle-ci. Cependant cette classe de base n'a pas un statut particulier et constitue une perspective au même titre que les autres perspectives.

- Les différentes perspectives sont complètement indépendantes et ne partagent aucune information. Chaque perspective regroupe les attributs propres à son espace de description.
- La description multiple est centralisée dans un unique état qui est celui de l'objet. Tout accès à une perspective s'effectue donc par un accès à l'objet.

Si l'approche de KRL pour la description multiple des objets a été retenue dans différents systèmes au début des années 80, elle est rarement décrite en détails. Par exemple, la hiérarchisation (possible, souhaitable?) des classes décrivant les perspectives ou d'éventuels partages de propriétés ne sont jamais abordés.

2.1.3. Points de vue et spécialisation simple

Le modèle de KRL nous permet de mettre en évidence une notion de point de vue liée à la spécialisation (simple). En effet, une *unit* peut comporter plusieurs perspectives différentes qui portent sur le même prototype. La perspective servant de base constitue un point de vue au même titre qu'une desription obtenue par spécilaisation. Cette notion est présente sur le plan conceptuel dans tout langage intégrant un mécanisme d'héritage. Par exemple, dans un langage de classes, où l'héritage est utilisé pour définir une classe Employé par spécialisation d'une classe Personne, i.e. Employé admet comme Personne super-classe directe, toute description des instances de Employé inclut une description de type Personne et tout employé est une personne. Cependant, accéder aux différents points de vue d'un objet dans un programme necessite l'utilisation de mécanismes spécifiques qui ne sont pas développés dans ce type de langages.

2.1.4. Points de vue et spécialisation multiple

Dans la proposition de KRL, on retrouve essentiellement l'idée des points de vue associée à la relation de spécialisation. On peut considérer des points de vue différents dès lors que l'on considère des spécialisations différentes et la notion de point de vue semble en conséquence liée à la spécialisation multiple. L'héritage multiple est un mécanisme qui permet la spécialisation multiple et on retrouve donc une notion similaire de point de vue dans tout système intégrant l'héritage multiple. En effet, les super-classes directes constituent autant de points de vue conceptuels selon lesquels on peut considérer une classe. L'héritage multiple offre une technique d'intégration des descriptions multiples par combinaison. Cela consiste à créer touts les combinaisons de classes possibles afin de prendre en compte tous les points de vue possibles. Ainsi, en considérant l'exemple ci-dessus, si toutes les facultés de l'université (Arts, Sciences Sociales et Sciences Pures) ont les trois types d'enseignants (Assistant, Professeur et Maître-conférence), le modèle aura une sous-classe pour chaque faculté et chaque grade comme présenté en figure 3.

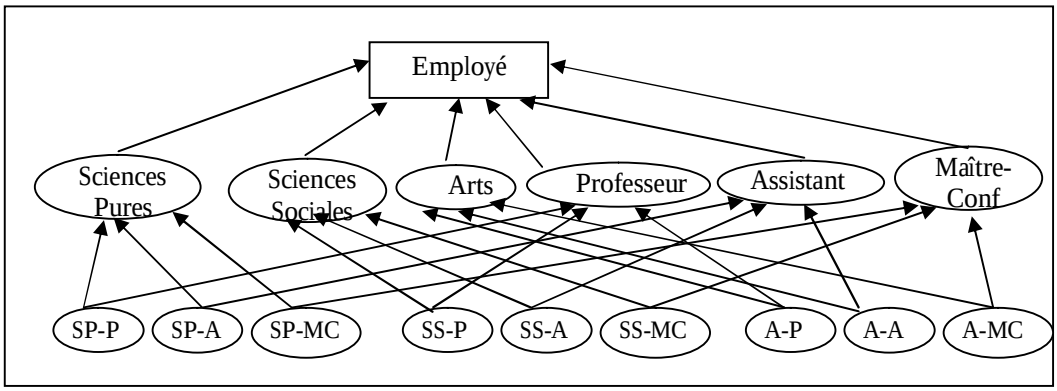


Figure 3. Modélisation des points de vue par l'héritage multiple

Cependant une telle représentation peut engendrer plusieurs problèmes liés à l'héritage multiple dont :

- Risque d'explosion combinatoire du graphe. Si, en plus, les facultés de l'université sont organisées en départements, le graphe de classes s'agrandit énormément.
- Dans la structure obtenue, les points de vue de départ sont mélangés; les mécanismes d'inférence de la base ne peuvent pas faire la distinction entre les classes appartenant à des perspectives différentes. De plus, rien n'empêche la création de la classe "Assistant-Professeur" qui n'aura jamais d'éléments car aucun employé ne peut être assistant et professeur à la fois.
- La représentation multiple est centralisée dans un seul état qui est celui de l'objet. La notion de point de vue est perdue car pour raisonner avec un tel objet, on est obligé de manipuler tous ses attributs et non pas seulement ceux d'un domaine d'intérêt.
- L'héritage multiple ne permet aucune évolution dynamique des objets puisque tout est figé dans le treillis des classes.

2.2. Points de vue et composition

Les objets composites sont des assemblages de sous-objets, leurs propres composants, qu'ils référencent par des attributs. Ils sont aussi appelés des agrégats. Les points de vue peuvent être considérés comme des formes d'objets composites interprétés comme des vues différentes sur une même entité conceptuelle. Dans ce qui suit, nous examinons à travers la présentation des perspectives dans le modèle LOOPS, la possibilité et les limites de l'utilisation des objets composites pour la prise en compte des points de vue.

2.2.1. Les perspectives dans le modèle LOOPS

LOOPS (Lisp Object and data Oriented Programming Language) [19] est un langage hybride qui fut l'un des premiers à intégrer la gestion d'objets composites. La notion d'objet composite est en particulier exploitée pour représenter les points de vue.

Contrairement à KRL, où tous les attributs sont regroupés dans le même objet, LOOPS divise l'instance même dans des objets différents, ses perspectives. Un objet et ses perspectives sont une sorte d'objet composite, les composants étant les différentes perspectives. Chaque perspective est un objet à part entière, indépendant, auquel on peut envoyer directement des messages de modification, d'élimination, etc. A ce titre, chaque perspective possède son propre espace de nom et de variables. Cette indépendance permet de définir des attributs de même nom ayant des sens différents dans plusieurs perspectives. L'idée originale du modèle LOOPS a été adoptée ultérieurement par d'autres approches [71] [16] [38].

Pour traiter les perspectives, LOOPS utilise deux classes abstaites appelées les mixins : *Node* et *Perspective.* Une classe qui décrit des objets qui sont des perspectives d'autres objets doit être définie comme sous-classe de la classe *Perspective,* et une classe qui décrit des objets ayant des perspectives est sous-classe de la classe *Node.* Notre exemple illustratif est traduit en LOOPS comme ci-dessous (Figure 4). Pour la clarté de la présentation, nous nous limitons aux deux perspectives de la classe Membre : Arts et Assistant.

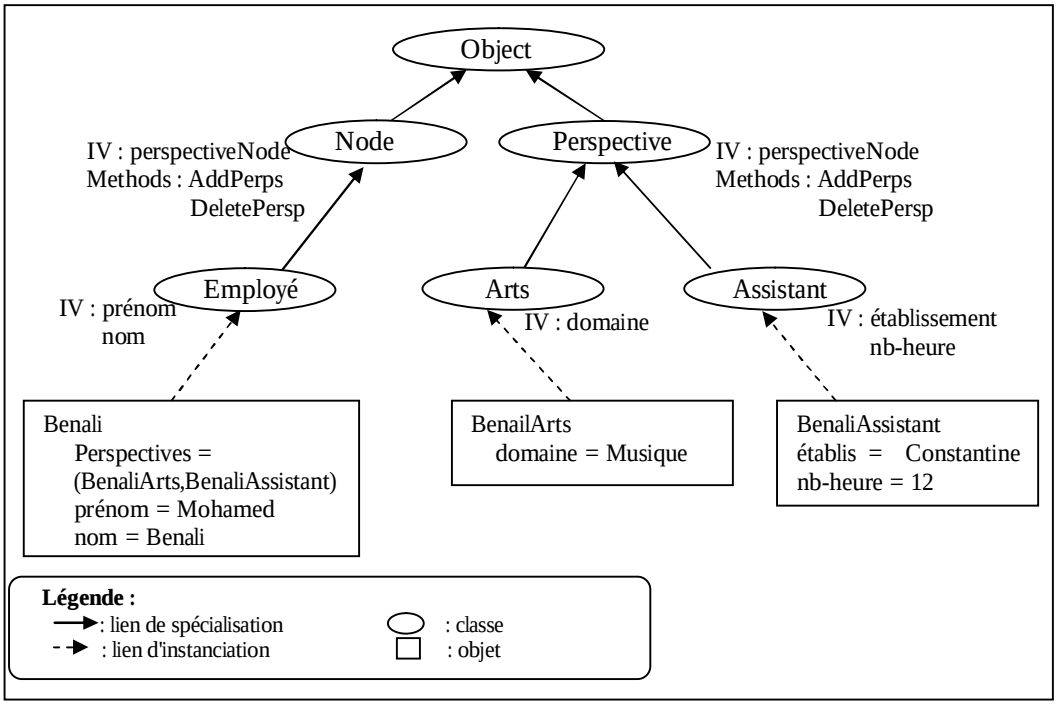


Figure 4. La représentation des perspectives par les objets composés dans LOOPS

Benali est un objet composite, instance de la classe Employé qui est sous-classe du mixin *Node.* Benali hérite la variable "perspectives" de la classe *Node.* Cette variable contient la liste des objets qui sont des perspectives de Benali : BenaliArts et BenaliAssistant, instances des sous-classes du mixin *Perspective.* Notons que les objets perspectives sont instanciés dynamiquement, i.e. il est possible d'ajouter ou de supprimer une perspective au cours de la vie d'un objet en utilisant les méthodes (AddPersp, DeletePersp) définies dans les maxins.

2.2.2. Spécificités de la représentation multiple dans LOOPS

D'après la présentation succincte de la technique de prise en compte des perspectives dans LOOPS, nous remarquons ce qui suit :

- LOOPS permet une représentation multiple décentralisée. En effet, la description d'une perspective n'est pas localisée au niveau de l'objet composite, à l'instar de KRL, mais dans un objet indépendant.
- L'accès à une perspective d'un objet est direct et ne nécessite pas l'accès préalable à l'objet composite.
- Dans la description de l'objet composite, les propriétés de base (prénom et nom dans notre exemple) ne sont pas considérées comme une perspective, comme c'est le cas dans KRL. Les propriétés de base sont inhérentes à l'objet et sont séparées des perspectives de celui-ci.
- Aucun partage de propriétés entre perspectives n'est signalé. En plus, une perspective ne peut pas accéder aux propriétés de base d'un objet.
- Les perspectives sont gérées dynamiquement à la demande.

2.2.3. Points de vue et objets composites

La modélisation des points de vue dans LOOPS apporte une nouvelle idée attrayante qui consiste à regrouper tous les points de vue d'une seule entité dans un seul objet composite. La notion d'objet composite existe dans la plupart des langages à objets. Dans un langage à classes standard, la composition est simplement exprimée à travers les variables d'instance. L'approche des perspectives dans LOOPS semble donc suggérer qu'une programmation par points de vue est possible dans tout langage à objets. Cette programmation doit simplement tenir compte d'une interprétation particulière des liens de composition.

A l'instar de ce qui vient d'être présenté, nous soulignons deux remarques importantes sur la notion d'objet composite telle qu'elle apparaît en programmation et la notion de perspective dans LOOPS.

- A l'exception de l'instanciation, les points de vue sont gérés de la même manière que les objets composites standards : les perspectives sont instanciées à la demande (un nouveau point de vue peut être ajouté dynamiquement à un objet si nécessaire) alors que les parties d'un objet composite sont toutes instanciées au moment de la création de celui-ci.
- L'interprétation des perspectives est radicalement différente de l'interprétation des parties d'un objet composite standard. En effet, toutes les perspectives d'un objet dénotent une seule et même entité du domaine, alors qu'un objet composite dénote une entité différente des entités dénotées par ses parties.

2.3. Points de vue et délégation

Parmi les premiers systèmes à fournir des possibilités proches de celles des points de vue en se fondant sur le mécanisme de délégation, nous citons le système PIE [43], le système SYSTALK [93] et l'approche des objets morcelés développée par Bardou

[10]. Bien que la première proposition fût décrite dans le système PIE, nous avons opté pour la présentation de l'approche de Bardou que nous jugeons plus significative.

2.3.1. Les objets morcelés : Approche de Bardou

C'est dans le cadre des langages à prototypes[3] que l'approche des objets morcelés a été proposée pour représenter les entités du monde réel selon plusieurs points de vue. Les langages à prototypes sont des langages de programmation sans classes, dans lesquels le problème de l'évolution dynamique des objets, appelés prototypes, ne se pose pas : un prototype possède sa propre description et peut donc la faire évoluer librement. Les langages à prototypes intègrent le clonage et la délégation comme mécanismes fondamentaux. Le clonage est un mécanisme de création d'objet. Il consiste à effectuer une copie d'un objet afin d'obtenir un nouvel objet ayant le même comportement et le même état. La délégation est un mécanisme de partage persistant de propriétés entre objets, au travers de liens généralement appelés liens de délégation ou encore liens "*parent*". Le mode opératoire de la délégation est décrit d'une manière informelle [55] : si un objet ne peut pas répondre à un message qui lui a été envoyé, alors le message est passé à son *parent* en suivant les liens de délégation.

Pour représenter des entités selon plusieurs points de vue, Bardou propose de rassembler les représentations d'une entité sous forme d'un objet unique ayant une identité propre appelé objet morcelé. Un objet morcelé est défini par une collection de *morceaux*. Ces morceaux n'ont pas le statut d'objet et ne sont pas instanciables. Cependant, chaque morceau est une unité d'abstraction qui détient les propriétés spécifiques à un point de vue particulier. Les morceaux sont organisés au sein d'un objet morcelé dans une hiérarchie de délégation (avec partage de propriétés). Cette hiérarchie admet une racine unique qui détient, éventuellement, les propriétés partagées par tous les points de vue sur un objet morcelé. Toute propriété détenue par un morceau est également accessible depuis les morceaux descendants.

La représentation de l'employé de notre exemple par un objet morcelé est illustrée dans la figure 5. On y trouve un seul objet, l'objet morcelé Benali dont la définition est divisée en trois morceaux nommés Employé, Arts et Assistant. Ces derniers sont organisés selon une hiérarchie de délégation qui dénote une relation de généralisation/spécialisation sur les points de vue de l'entité représentée. Employé est le morceau qui représente le point de vue le plus général sur l'objet Benali.

[3] Les langages à prototypes sont inspirés par la théorie des prototypes développée en sciences cognitives et qui a notamment donné naissance aux langages de frames en représentation des connaissances.

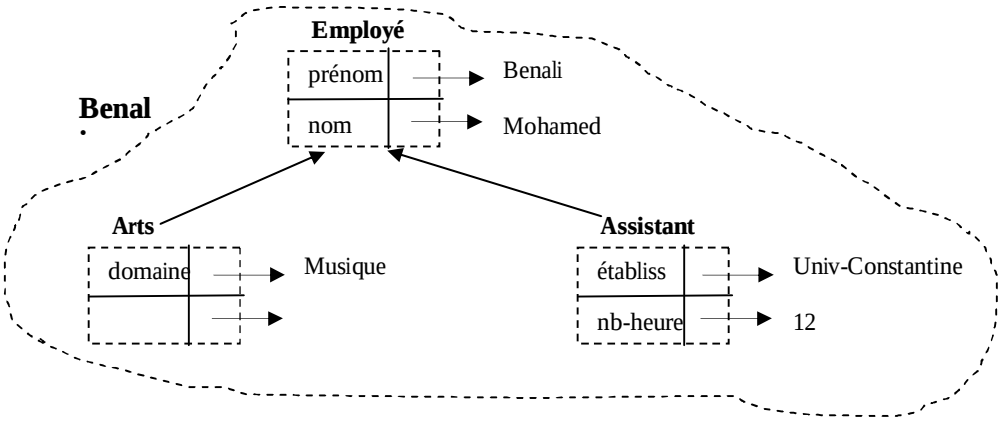


Figure 5. La représentation d'une entité par un objet morcelé

Plusieurs opérations élémentaires peuvent être définies pour manipuler les objets morcelés dont le nommage et l'accès, la création et la copie, la modification d'objets morcelés [10]. L'activation des propriétés (exécution d'une méthode, accès à une variable) d'un objet morcelé se fait par envoi de messages. Seul l'objet morcelé, ayant le statut d'objet, peut recevoir des messages. Le nom du morceau dénotant le point de vue doit être spécifié dans le message. Le sélecteur de la propriété est alors recherché dans le morceau désigné puis dans ses ascendants en cas d'échec.

2.3.2. Spécificités de la représentation multiple dans l'approche de Bardou

L'idée originale de la représentation multiple par des objets morcelés consiste en la possibilité de partage de propriétés entre les points de vue via la relation de délégation, mécanisme des langages à prototypes par excellence. Les spécificités d'une telle représentation se résument aux points suivants :

- La représentation multiple est centralisée au niveau de l'objet morcelé qui est le seul à posséder le statut d'objet.
- Les points de vue (les morceaux) n'ont pas une identité propre. Ils ne peuvent pas être désignés et n'ont pas de raison d'être en dehors du contexte de l'objet morcelé.
- Les morceaux d'un objet morcelé ne sont pas complètement indépendants. Ils sont liés par une relation de généralisation/spécialisation dénotée par la hiérarchie de délégation qui induit un partage de propriétés.
- Les objets morcelés peuvent être manipulés dynamiquement en leur ajoutant ou supprimant des morceaux.

2.3.3. Points de vue et objets morcelés

L'approche proposée par Bardou a pour objectif l'intégration de la notion de point de vue dans les langages à prototypes. Dans son étude, Bardou s'est principalement intéressé à préciser la conséquence, en termes de partage, de l'application du mécanisme de délégation et à mettre en évidence le rapport entre ce mécanisme et la notion de point de vue. Cette étude a amené à proposer les objets morcelés, dans lesquels la délégation est utilisée, pour obtenir une représentation multiple.

Cependant, bien qu'issus d'une étude sur les langages à prototypes, les objets morcelés peuvent être intégrés dans un langage de classes en utilisant l'héritage de classes comme mécanisme de partage de propriétés. En effet, la délégation et l'héritage de classes représentent deux mécanismes de partage de propriétés et ont un mode opératoire très similaire. Dans les deux cas, l'application du mécanisme consiste généralement à effectuer, lors d'un envoi de message, une recherche du sélecteur en suivant les liens d'héritage. La différence, par contre, réside dans le niveau d'abstraction auquel la relation d'héritage intervient : l'héritage intervient directement au niveau des objets dans les langages de prototypes avec délégation alors que l'héritage de classes est défini entre les descriptions d'objets, c'est à dire les classes, dans les langages à classes. Plusieurs suggestions ont été faites pour l'intégration des objets morcelés dans un langage de classes. Celles-ci proposent de morceler au niveau des classes la description des objets. Le modèle de données orienté rôle développé dans [30] repose sur le morcellement des classes pour la représentation des objets à rôles multiples.

3. Points de vue en programmation

L'intégration de la notion de vue/point de vue dans les langages de programmation à objet a fait l'objet de plusieurs travaux de recherche et réalisations. Les relations classiques de l'approche objet (héritage, délégation, composition) sont utilisées pour implanter cette notion. Parmi ces travaux, nous citons : la programmation par sujets [46], la programmation par aspects [49], la programmation par vues [59] [62] et la programmation par objets structurés en contextes de CROME [91] [32]. Nous présentons les deux approches de programmation par vues et objets structurés en contextes.

3.1. Programmation par objets structurés en contextes

La programmation par objets structurées en contextes est issue des notions de points de vue du modèle de représentation des connaissances ROME (Représentation d'Objets Multiple et Evolutive) [25] [26] [33]. ROME remet en cause la contrainte de la représentation unique et figée des objets induite par le mécanisme classique d'instanciation en proposant deux principes de base :

- Le principe de la représentation multiple qui permet à un objet d'être représentant direct de plusieurs classes.
- Le principe de la représentation évolutive qui permet à un objet d'évoluer à travers la hiérarchie de classes.

Un objet à représentation multiple et évolutive dans ROME est dénoté par r-objet et possède une seule classe d'instanciation qui décrit ses caractéristiques de base et peut être rattaché par un lien dynamique de représentation (et non pas un lien d'instanciation) à une ou plusieurs classes dites de représentation. Une classe de représentation est une sous-classe d'une classe d'instanciation et permet d'affiner la description de celle-ci. Une classe d'instanciation d'un r-objet permet de limiter le treillis de représentation potentiel à l'intérieur duquel celui-ci peut évoluer. Un lien de représentation est traduit par une représentation de l'objet selon un point de vue. Le concept de point de vue est utilisé dans ROME pour résoudre les problèmes de conflits causés par l'héritage multiple.

La figure 6 présente un exemple de représentation dans ROME. L'objet P est une instance de la classe d'instanciation Membre et possède deux liens de représentation avec les classes Maitre-Conf et Sportif.

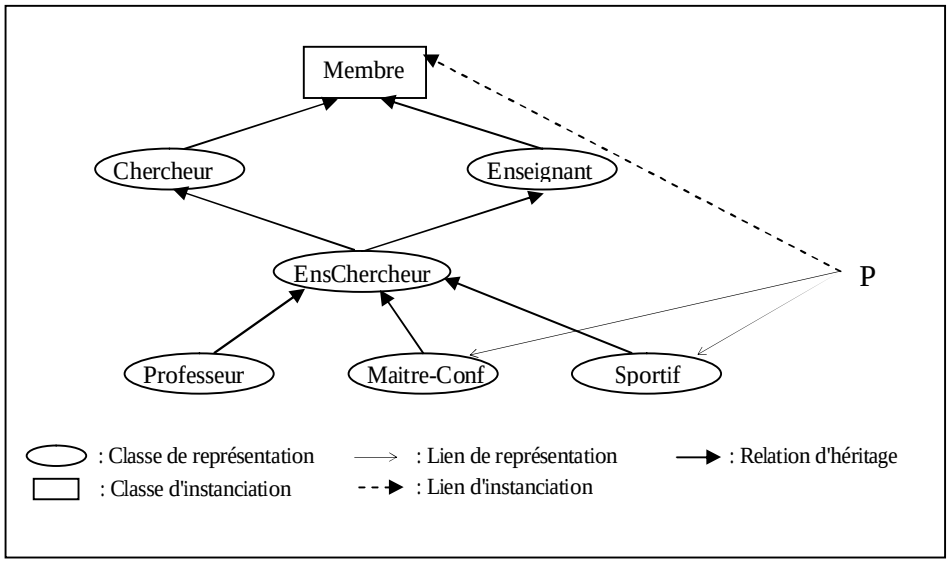


Figure 6. La représentation multiple dans ROME

Transposé dans le monde de la programmation, le modèle ROME a donné naissance à la programmation par objets structurés en contextes : CROME (Contextes en ROME). CROME propose un cadre de programmation qui améliore la modularité des systèmes par des contextes en se basant sur une structure en plans. On distingue deux types de plans : le plan de base et des plans fonctionnels.

- Un plan de base permet d'identifier un référentiel d'objets qui constitue les objets de base d'un système. Le plan de base procure à ces objets une première description qui peut être plus ou moins riche selon le niveau de détail souhaité. Les objets sont décrits dans le plan de base par des relations d'héritage traditionnel.
- Les plans fonctionnels permettent des enrichissements et des extensions spécifiques des classes du plan de base en ajoutant des attributs et des méthodes. Ils n'introduisent pas de nouveaux objets mais étendent la description des objets du référentiel pour répondre aux spécificités des différents contextes du système.

Un contexte en CROME est l'association d'un plan fonctionnel et du plan de base. C'est un programme qui implique les objets du système pour accomplir une fonction particulière selon un point de vue donné.

Notons que les classes d'un plan fonctionnel ne sont pas instanciables mais préservent l'identité des objets de base, à l'instar des classes de représentations dans ROME. Chacune de ces classes hérite systématiquement de la classe de base qu'elle étend. Les relations d'héritage des classes du plan de base sont préservées dans chaque plan fonctionnel.

La figure 7 traduit l'exemple simple de la figure précédente dans le contexte de CROME. Le plan de base comprend la classe "Membre". Celle-ci est partagée par les différents plans fonctionnels qui sont : le plan "recherche" et le plan "enseignement".

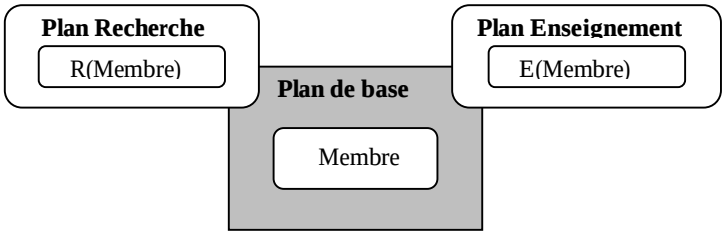


Figure 7. Plan de base et plans fonctionnels dans CROME

L'enrichissement d'une classe de base pour une fonction donnée est appelé "partie fonctionnelle". Une partie fonctionnelle est identifiée par le nom du plan fonctionnel et la classe de base dont elle dérive, exemple Recherche(Membre). Dans une partie fonctionnelle, aucune restriction n'est imposée pour l'appel de méthodes d'autres parties fonctionnelles de la même classe. La communication inter-contextes est donc tolérée.

La programmation en CROME est une approche modulaire qui propose une nouvelle forme de réutilisation autre que l'héritage et la composition par l'élaboration de parties fonctionnelles. Cependant, elle présente quelques limites notamment le manque d'une démarche pour identifier les contextes d'un système.

3.2. Programmation par vues

La programmation par vues est issue des travaux de Mili et al [59] [60] [62] sur les systèmes d'informations d'entreprises. C'est une approche de programmation qui exploite le constat que dans un système d'informations, les mêmes entités jouent des rôles fonctionnels différents. Ces rôles correspondent souvent à des processus génériques qui ne dépendent pas du domaine d'affaires. La notion de vue est donc introduite et qui consiste en une construction logicielle réalisant un rôle fonctionnel. On s'intéresse principalement aux problèmes de développement, de réalisation, de la maintenance et du déploiement de ces vues.

Le principe de la programmation par vue repose sur la représentation d'un objet par un "objet de base" et un ensemble variables de vues représentant des facettes fonctionnelles. Chaque vue possède des propres données et son propre comportement et pouvant accéder aux données et services de l'objet de base. Le partage de variables et du comportement sont réalisés par le mécanisme de délégation et le mécanisme de redirection de message, respectivement.

La logique inhérente aux vues peut être perçue comme une instanciation d'une logique plus générique, qui correspond à un processus générique. Par exemple, dans un service de comptabilité on peut considérer un avion, une voiture, un ordinateur, etc. des pièces d'équipement ayant un prix d'achat et peuvent être amorti sur une période de temps. Ce "comportement comptable" pourrait être codé de façon générique et instancié par différents objets. La figure 8 présente un exemple d'une vue générique *CapitalAmortissable* attachée à un objet générique de base *Matériel* spécifié dans la clause REQUIRES. Le point de vue *CapitalAmortissable* décrit les primitives

additionnelles concernat l'amortissement de matériel dans la clause PROVIDES. La clause EXTENDS est utilisée pour spécifier des blocs de code, exécuté par la vue, avant (before) et (after) des méthodes de l'objet de base. Par exemple, la méthode de l'objet de base *setDateAchat* est exécutée avant la méthode de la vue *setPériodeAmortissement*.

```
VIEWPOINT CapitalAmortissable
    REQUIRES Matériel
        EXTENDS
        void setDateAchat (Date d)
            after
            { setPériodeAmortissement (Year (Date : : today ( ) - d) ; }
    PROVIDES
        variables
            Year _périodeAmortissement ;
            TypeMonnaie _valeurRésiduelle ;
        opérations
            TypeMonnaie getValeurResiduelle ( ) { return _valeurRésiduelle ; }
            void setPériodeAmortissement (Year y) {-périodeAmortissement = y ; }
            Date getAge ( ) {return Date : : today ( ) – getDateAchat ( ) ; }
END VIEWPOINT
```

Figure 8. Exemple de point de vue (tiré de [61])

En C++, un point de vue est instancié selon la syntaxe :

Defview FCamion as CapitalAmortissement [Camion]

Où *Camion* est une instance de l'objet de base *Matériel*. Cette instanciation applique le point de vue *CapitalAmortissement* à la classe de base *Camion*. Le résultat est une classe (vue) *FCamion*. La figure 9. Illustre la classe générée FCamion. Cette dernière a comme attribut et méthodes ceux déclarés dans la claude PROVIDES du point de vue *CapitalAmortissement*. Si une méthode de *FCamion* fait référence à un attribut de l'objet de base, une transformation est faite de façon à tenir compte de la délégation. Par exemple, la méthode "Date getAge( )" qui fait partie de la clause PROVIDES est transformée dans la clause *FCamion* comme suit :

Date FCamion : : getAge ( ) {

Return Date : : today ( ) – camion ->getDateAchat ( ) ;}

```
class FCamion {
public :
FCamion (TypeID unId );
…
Float getValeurRésiduelle ( ) ;
void setPériodeAmortissement (float) ;
Date getAge ( ) ;
private :
Camion *-camion ;
Year _périodeAmortissement ;
typeMonnaie valeurRésiduelle ;
… }
```

Figure 9. La classe FCamion

La notion de point de vue dans la programmation par vues présente un double objectif : d'une part elle permet la réutilisation des vues et d'autre part offre un moyen pour découper le développement des vues de celui des objets de base.

4. Points de vue dans les ateliers et les environnements de génie logiciel

En génie logiciel et dans la modélisation du processus de développement, la notion de point de vue a eu beaucoup d'intérêt. La nécessité d'utiliser les points de vue multiples lors de la modélisation et du développement des systèmes complexes et hétérogènes s'est exprimée dans différents travaux de recherche. Nous présentons dans ce qui suit succinctement la méthode SADT [76] et l'approche de [39] [40] [69].

4.1. SADT

La méthode de modélisation SADT (Structured Analysis ans Design Technique) [76] repose sur le principe que la spécification d'un modèle est fortement liée à la façon dont on le voit. SADT est une méthode de modélisation des données et des activités d'un système, descendante, modulaire, hiérarchique et structurée. Elle impose au modélisateur de choisir son point de vue dès le départ et de spécifier son système entier selon ce point de vue. Les éléments décrits dans les différents modèles SADT sont identiques, mais la pertinence de chacun d'eux, la terminologie utilisée et le niveau de détail nécessaire sont différents. Le point de vue représente donc la perspective dans laquelle le problème est étudié.

Dans SADT, spécifier un problème selon plusieurs points de vue revient à obtenir plusieurs modèles indépendants, chacun est fortement lié au point de vue choisi. Construire une maison, par exemple, peut être décrit selon le point de vue du maçon, du charpentier, du promoteur, etc. et donne lieu à des modèles différents puisque les activités sont présentées de façon différente dans chaque modèle et que les données communes ne se retrouvent pas au même niveau de détail.

Cependant, l'inconvénient majeur de cette méthode est qu'elle ne propose aucun mécanisme de corrélation entre les différents modules, et se confronte à une modélisation multimodèle cloisonnée.

4.2. VOSE

VOSE (ViewPoint-Oriented Systemd Engineering) [69] est un environnement de développement des systèmes hétérogènes, complexes et composites qui requièrent l'utilisation de notations et de stratégies de développement multiples pour décrire les perspectives de différents développeurs (participants au processus de développement). VOSE est l'implémentation de l'approche proposée par Finkelstein et al. [39] [40] [69].Cette approche utilise les points de vue pour décrire les participants au développement d'un système, leur rôle dans le développement et leur perception du domaine. Un point de vue est défini comme "un objet faiblement couplé, géré localement, encapsulant les connaissances sur les modes de représentation, sur le processus de développement et sur les spécifications partielles d'un système et de son domaine" un point de vue est une unité encapsulant la connaissance de développement

d'un système et est composée de cinq parties appelées *slots* : un *style* ou le formalisme de représentation ; un *plan* qui décrit les actions, le processus et les stratégies de développement ; le *domaine* qui détermine un champs d'intérêt particulier par rapport au système global ; la *spécification* qui est la description d'un domaine par un style, et *l'historique* qui garde une trace des actions exécutées dans le processus de développement.

Un point de vue est associé à un développeur particulier appelé son *Owner*. Ce dernier est responsable d'activer les actions et les stratégies de développement du slot *plan* pour fournir une *spécification*, suivant le formalisme de représentation décrit dans le slot *style*, pour sa connaissance du domaine. De ce fait, plusieurs points de vue peuvent utiliser le même style et le même plan pour produire différentes spécifications pour différents domaines d'où l'introduction du concept *template*. Un *template* est un type de point de vue composé des deux slots : *style* et *plan*. Un point de vue est donc une instance d'un template (voir figure 10).

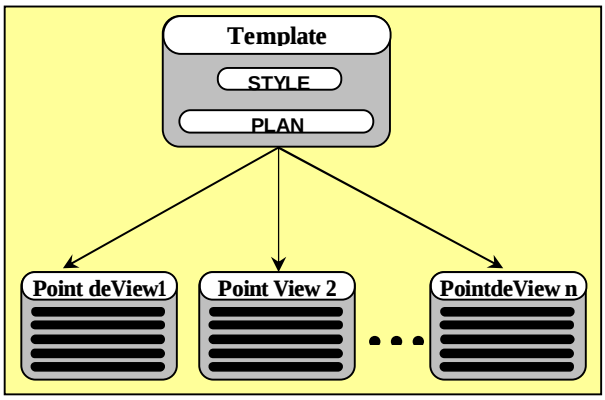


Figure 10. Instanciation d'un template par plusieurs points de vue

Dans cette approche une méthode de développement est une collection de templates. Une méthode de développement par points de vue d'un système consiste alors en l'instanciation de ces templates pour les différents domaines du système et en la gestion des différents types de corrélation inter-points de vue. Notons que ces derniers sont spécifiés au niveau du slot plan.

4.3. Discussion

Dans les approches de modélisation du processus de développement, le paradigme de point de vue n'est pas exploité au niveau de la description des entités d'un domaine d'étude mais plutôt au niveau de la gestion des connaissances du processus de développement et de la gestion des fragments de spécification. Ces approches permettent de :

- Mettre l'accent sur les différentes expertises concernant le métier et le processus de développement,
- Se baser sur la corrélation inter-points de vue en proposant des mécanismes et des méthodes d'intégration des fragments de spécification.

5. Conclusion

Dans ce chapitre, nous avons fait mention d'un certain nombre de travaux qui ont abordé la notion de points de vue dans différents domaines tels que la représentation des connaissances, le génie logiciel, la programmation et la modélisation des systèmes complexes. Dans le cadre de notre thèse, nous nous intéressons plus particulièrement à son intégration dans le domaine des bases de données à objets que nous avons omis son abord dans ce chapitre. En fait, et pour la clarté de l'exposé nous en consacrons le chapitre suivant afin de pouvoir apporté d'amples détails.

Que les points de vue aient été appelés points de vue, perspectives, rôles, vues, contextes ou aspects, ils découlent en général de la possibilité de considérer dans un même domaine, modèle, ou système des descriptions ou des représentations multiples de la même "chose"[4]. Parmi les notions apportées par les différentes approches d'intégration des points de vue, les concepts qui nous semblent essentiels à retenir sont : la possibilité de construire des points de vue à partir d'un modèle de référence unique, la manière de les organiser, la capacité de leur évolution dynamique et la gestion de la cohérence du modèle global. Définir un mécanisme de point de vue supportant tous ces aspects devrait permettre d'atteindre une modélisation des systèmes complexes de qualité.

[4] La nature de cette "chose" peut varier suivant le domaine, le système ou le modèle.

Chapitre 2

Où l'on présentera quelques travaux intégrant la notion de point de vue dans le domaine des bases de données. En effet, le concept de point de vue a été essentiellement étudié dans le cadre des travaux de recherche menés sur la notion de vue et celle de rôle. Plusieurs modèles de données ont été proposés. La plupart de ces derniers sont fondés sur un modèle à classes, en descendance directe des langages de programmation par objets du même nom. Il est indéniable que ce modèle et les concepts associés (encapsulation, héritage, polymorphisme) permettent la construction de modèles plus simples à comprendre, à valider, à faire évoluer et à réutiliser. Cependant "Le monde n'est ni plat ni séquentiel, mais au contraire multidimensionnel et hautement parallèle". D'où l'insuffisance de ces concepts pour la prise en compte de nombreux aspects du monde réel en occurrence la représentation multiple des objets et l'évolution de leur structure et de leur comportement.

En effet, le modèle objet fournit un processus de conception guidé par les données[1]*. Il peut être retrouvé sous diverses variations, dans différents langages et systèmes, mais il peut être globalement caractérisé dans sa forme la plus courante par les points suivants :*

- *L'identité : tout objet possède une identité propre qui est uniforme et indépendante de son contenu.*
- *La classification : tout objet est crée par instanciation d'une classe. Celle-ci décrit la structure et le comportement de ses objets. Un objet, instance d'une classe doit se conformer à la description de sa classe d'appartenance.*
- *Le polymorphisme : un traitement peut avoir des interprétations différentes selon la classe de l'objet auquel on applique.*
- *L'héritage : une classe peut être définie par spécialisation d'une autre classe (alors appelée sa super-classe), grâce au mécanisme d'héritage. L'héritage permet de promouvoir l'extensibilité et la réutilisation des classes en les liant par une hiérarchie taxonomique.*

Le modèle à classes offre plusieurs avantages largement reconnus en programmation, parmi lesquels on peut citer la liaison dynamique, l'encapsulation et un grand pouvoir d'abstraction et de réutilisation basé sur l'héritage entre les classes. Néanmoins, il comporte plusieurs limitations directement liées à la dépendance forte qui existe entre un objet et la classe dont il est instance. Parmi ces limitations, nous citons :

- *Un objet doit toujours se conformer à la description fournie par sa classe d'instanciation. Sa structure et son comportement ne peuvent pas évoluer dynamiquement. La reclassification dynamique autorisée dans certains langages, tel SMALLTALK, ne suffit pas à résoudre ce problème, un objet ne peut pas en tout état de cause évoluer à titre individuel.*
- *En tant qu'instance d'une seule classe, la description (structure et comportement) d'un objet est unique et rien n'est à priori prévu dans le modèle pour la considérer selon plusieurs points de vue.*

---

[1] Contrairement au modèle relationnel qui est un processus de conception guidé par les valeurs.

Ces deux problèmes ne sont pas complètement indépendants l'un de l'autre. Si l'on dispose d'un moyen de considérer un objet selon plusieurs points de vue, alors une solution au premier problème consiste à associer un point de vue différent à chaque stade de l'évolution d'un objet. Inversement, si l'on peut faire évoluer dynamiquement le comportement et la structure d'un objet, en lui appliquant les transformations adéquates et en mémorisant ses différentes versions, on peut obtenir plusieurs points de vue dans la représentation d'un objet.

Dans ce chapitre, nous présentons quelques modèles qui ont été développés en bases de données objets pour la prise en compte des vues et des rôles. Puis, nous comparons ces notions avec celle des points de vue considérée dans le cadre de notre étude.

1. Les vues

La notion de vue est initialement conçue pour décrire des schémas externes qui permettent la définition de sous-schémas personnalisés du schéma de la base de données d'origine [6]. Les opérateurs algébriques tels que la sélection, la projection ou la jointure, permettent la construction de vues différentes qui permettent la représentation des données selon différents points de vue et en fonction d'objectifs spécifiques à diverses applications.

La notion de vue a été utilisée pour résoudre des problèmes liés à la configuration de l'interface utilisateur, la protection des données, l'évolution des besoins d'organisation des données sans perturbation de l'existant et l'optimisation des requêtes [3] [36] [89]. Actuellement l'utilisation des vues s'est étendue aux applications distribuées. Le mécanisme d'import/export des vues [81] a été exploité dans de nombreux travaux pour assurer l'intégration et l'interopérabilité des systèmes hétérogènes et distribués.

Dans les bases de données à objets, il n'y a pas de consensus sur la définition de la notion de vue. On enregistre autant de définitions que de systèmes. Cependant, nous pouvons distinguer deux tendances principales. Dans la première, une vue correspond à une classe dérivée par une requête à partir d'une ou de plusieurs classes de la base de données. La seconde tendance, selon plusieurs travaux [47] [88] [2] [17] [38], considère une vue comme un schéma virtuel dérivé à partir d'un schéma de la base de données [66] [65]. Elle est appelée ainsi car elle réfère à des données non persistantes mais calculées. Nous présentons ici deux systèmes intégrant le concept de vue dans les bases de données à objets puis nous comparons celui-ci avec la notion de point de vue considérée dans le cadre de notre étude.

1.1. COCOON

COCOON [79] [80] est un modèle d'objets pour les bases de données dans lequel la distinction est faite entre les types et les classes. Les types sont des spécifications d'interfaces alors qu'une classe est un conteneur d'objets d'un certain type. Un objet COCOON est l'instance d'un type de données abstrait que l'on ne peut manipuler que via son interface. Les types sont organisés dans une hiérarchie de sous-typage dont la racine est *Object*. Les classes sont définies comme des ensembles typés d'objets : elles ont une extension dans laquelle on ne trouve que des instances du même type, ou de

sous-types de celui-ci. Les classes et les types sont complètement séparés, et plusieurs classes peuvent avoir le même type. Les classes sont organisées dans une hiérarchie, dans laquelle deux classes sont liées par la relation "sous-classe-de" à condition que l'extension de la première soit un sous-ensemble de l'autre, ou que son type soit un sous-type de l'autre, ou encore que ces deux conditions soient vérifiées en même temps. Parmi les concepts et les mécanismes exploités dans COCOON nous citons : l'instanciation multiple (*multiple instanciation*) par laquelle un objet peut être instance d'un ou de plusieurs types qu'il peut acquérir ou perdre dynamiquement, l'appartenance multiple aux classes (*multiple classe membership*) qui autorise un objet à être membre de plus d'une classe et le concept des prédicats de classe (*class predicates*) qui présentent des conditions nécessaires et parfois suffisantes qui permettent une classification automatique des objets des superclasses dans leurs sous-classes.

La figure 1 montre la création de types et de classes dans COCOON. ObjectT[2] est le type racine dans la hiérarchie des types d'une base de données. PersonneT est un sous-type de ObjectT. AdolescentC est une sous-classe de la classe PersonneC ayant le même type. Cependant, l'ensemble des éléments de AdolescentC est un sous-ensemble des celui de sa superclasse PersonneC. Le prédicat de classe "age <18" spécifié dans AdolescentC est une condition nécessaire mais pas suffisante aux objets de la classe PersonneC pour être membres da la classe AdolescentC. La modification du mot clé **some** par **all** rend cette condition nécessaire et suffisante et permet une **classification automatique** par le système de tout objet de PersonneC vérifiant cette condition dans AdolescentC.

```
type PersonneT is_a ObjectT =
     nom : string,
     prénom : string,
     age : integer,
     …;
type EmployéT is_a PersonneT =
     salaire : real,
     nb-enfant : integer,
     …..;
class PersonneC : PersonneT some ObjectC;
class AdolescentC : PersonneT some PersonneC where age<18 ;
class EmployéC : EmployéT some PersonneC;
```

Figure 1. La définition de types et de classes dans COCOON

1.1.1. Les vues dans COCOON

Dans COCOON les classes du schéma global sont appelées les classes de base et leurs objets sont appelés objets de base. La notion de vue est étroitement liée à celle de classe et de requête. Toute vue est le résultat de l'évaluation d'une requête. Cette dernière porte sur d'autres classes existantes dans le système (classes de base ou classes définies par des vues). Les requêtes sont spécifiées en utilisant des opérations similaires à celles habituellement retrouvées dans l'algèbre relationnelle, telles que la sélection, la différence, l'union, la projection, etc. Cependant, le langage de requêtes utilisé dans COCOON repose sur une algèbre orientée-ensemble (*set-oriented algebra*) dont le principe est la préservation des objets (*object-preserving*): le résultat de toute requête

[2] Les noms de types, de classes de base et de vues sont suffixés pat la lettre T, C et V respectivement.

est un ensemble d'objets de base. Ce principe est d'une importance cruciale dans la propagation automatique des mises à jour.

A la création d'une vue, son type est automatiquement déterminé par le système (grâce aux informations relatives aux types des classes utilisées dans la requête) et son extension est calculée afin de la placer dans la hiérarchie des classes.

Soit à définir, à partir de la base présentée en figure 1, la vue JeuneC représentant les jeunes personnes de moins de 30 ans. Cette vue sera exprimée par une requête avec un opérateur de sélection comme suit :

define view JeuneV **as select** [age <30] (PersonneC)

Le résultat consiste en la création d'une nouvelle classe JeuneV, sous-classe de PersonneC, contenant les objets qui satisfont le prédicat de la sélection. Le type de la nouvelle classe (la vue) est celui de la classe de base dont elle dérive, i.e. la classe PersonneC.

La figure 2 montre le positionnement de cette classe dans la hiérarchie des classes et qui est assuré automatiquement par le système.

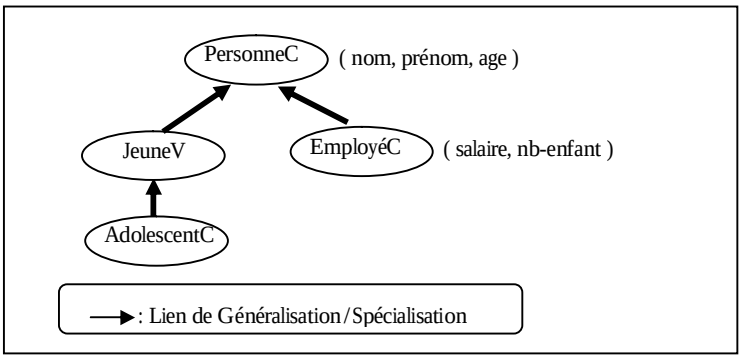


Figure 2. La création d'une vue dans COCOON

1.1.2. Mise à jour des vues

Les vues sont maintenues par calcul, contrairement aux classes de base dont l'extension est stockée. Elles peuvent être mises à jour, ou être utilisées dans la définition d'autres vues à l'instar des classes de base. Le principe de la préservation des objets par les vues permet la propagation de toute modification de la vue vers les objets des classes de base. Prenons l'exemple de la définition d'une vue par sélection, soit la vue JeuneV définie ci-dessus. Toute opération d'insertion, ajout, destruction ou suppression d'un objet dans JeuneV est directement effectuée sur la base des objets de la classe PersonneC et vice versa. Ainsi, afin de garantir l'intégrité suite à des mises à jour, le système procède à une re-classification automatique des objets : la définition d'une vue par sélection consiste en la création d'une classe dérivée V, sous-classe de sa classe de base B par un prédicat P. Cela correspond exactement à la création d'une classe dans le schéma de la base de données (B et V ont le même type T) comme suit :

Define class V : T **all** B **where** P;

La mise à jour d'une vue est définie de telle façon que seule son extension soit modifiée, les objets d'une vue n'étant pas modifiés ni crées, mais simplement enlevés ou ajoutés à son extension. Cela permet d'assurer l'indépendance des vues vis-à-vis des

données stockées et permet la définition de plusieurs vues indépendantes sur la même base. Plusieurs vues peuvent enfin être regroupées dans un sous-schéma (un sous-schéma du schéma de la base). L'utilisation combinée des sous-schémas et des vues permet de modifier le schéma d'une application, indépendamment des schémas d'autres applications de la même base [44].

1.1.3. Caractéristiques du modèle de vue dans COCOON

Le mécanisme des vues dans COCOON se caractérise par :

- Il repose sur le principe de la préservation des objets par lequel l'extension d'une vue est considérée comme une collection d'objets dérivés à partir des classes de base. La génération de nouveaux objets n'est donc pas prévue ni permise.
- Le modèle autorise l'instanciation multiple et l'appartenance multiple des objets aux classes. Ces deux caractéristiques découlent directement du principe de la préservation des objets. Par exemple, dans la création d'une vue par projection, l'ensemble des objets résultat est le même que celui de la classe de base mais avec un type différent.
- La re-classification dynamique des objets après des opérations de mise à jour. La modification d'un objet peut engendrer sa reclassification dans une classe plus spécifique (par exemple il satisfait maintenant les prédicats d'une classe) ou dans une classe plus générale (si un prédicat de sa classe d'appartenance est violé par l'opération de mise à jour). Ce qui offre une flexibilité de la dynamique des objets.

1.2. MultiView

MultiView [77] [53] est un système de gestion des vues construit sur le système de gestion de bases de données à objets GEMSTONE. MultiView introduit la notion de **classe virtuelle** et de **schéma virtuel** pour la spécification des **schémas vue**. Un langage de requêtes inspiré de l'algèbre relationnelle (sélection, différence, projection, union) est proposé pour la dérivation des classes virtuelles à partir des classes définies dans le schéma de la base ainsi qu'une méthodologie pour la spécification des vues[3].

1.2.1. Classe de base, classe virtuelle et schéma vue

MultiView distingue entre les **classes de base** et les **classes virtuelles**. Une classe de base est créée au moment de la définition initiale du schéma de la base de données. Une classe de base détient un ensemble persistant d'objets (son extension). Une classe virtuelle, par contre, est définie par application d'une requête et ajoutée dynamiquement au schéma de la base de données. Toutes les classes, qu'elles soient de base ou virtuelles sont organisées au sein d'une même hiérarchie de spécialisation contenue dans le **schéma global** du système.

Alternativement, au schéma global, d'autres schémas peuvent être considérés parmi lesquels on distingue le **schéma de base** qui contient toutes les classes de base, et les **schémas de vue** qui contiennent un sous-ensemble des classes (de base ou virtuelles)

[3] Employé au sens strict, le terme "vue" désigne un schéma de vue en MultiView, mais il est cependant souvent employé par abus de langage pour désigner une classe virtuelle.

du schéma global. Le schéma global étant constitué par une hiérarchie de spécialisation, on peut considérer un schéma de vue comme étant un sous-graphe du schéma global. Plusieurs schémas de vue peuvent être définis sur la même base de données (donc sur le même schéma global) et proposent autant de points de vue différents sur celle-ci.

1.2.2. Méthodologie du modèle des vues dans MultiView

MultiView propose une méthodologie pour le support des schémas vue multiples constituée de trois étapes indépendantes :

1. la dérivation des classes virtuelles à partir des classes définies dans le schéma global via des requêtes orientées objet,
2. l'intégration des classes virtuelles dans un même schéma consistant appelé le schéma de base,
3. la spécification de schémas vues complexes dérivés à partir du schéma global étendu. Un schéma vue est constitué de classes de base et de classes virtuelles.

Dans la première étape, les classes virtuelles permettent l'adaptation des classes du schéma global à des besoins spécifiques en modifiant la description des objets, en limitant l'accès à un ensemble restreint d'objets, en partitionnant les objets dans des classes adaptées à une application donnée, etc. La deuxième et la troisième étape sont des processus automatisés dont les algorithmes proposés permettent la classification et l'intégration des classes virtuelles et la construction de schémas vue consistants, respectivement. Cependant, la classification automatique peut, en conséquence, entraîner la restructuration des classes et des objets en créant, éventuellement, des classes intermédiaires pour le seul besoin de classification. En plus, le maintien du principe d'unique point d'héritage (*single point of inheritance*) sur lequel repose le mécanisme d'héritage et selon lequel deux classes ne peuvent partager une définition de propriété commune que si celle-ci est héritée depuis une plus petite super-classe commune, peut engendrer la migration de propriétés d'une classe à l'autre se répercutant en une migration de valeurs au niveau des objets. Cela alourdit considérablement l'emploi des taxinomies et peut mener à l'explosion de la hiérarchie globale avec des classes artificielles qui n'ont pas de raison d'être au niveau conceptuel.

La figure 3 présente un exemple illustrant la méthodologie du modèle MultiView pour supporter les schémas de vue. Dans la figure 3.(a), un schéma de base est défini. Deux classes virtuelles *Mineur* et *Adolescent* sont dérivées par une requête qui sélectionne des objets de la classe de base *Personne* dans la figure 3.(b). Après la création d'une classe virtuelle, celle-ci est intégrée automatiquement dans le schéma de base grâce à un algorithme de classification, qui en comparant les types et les extensions des différentes classes, trouve la place appropriée dans le schéma. Le résultat de l'intégration des classes virtuelles *Mineur* et *Adolescent* est présenté en figure 6.(c). La création d'un schéma vue passe par deux étapes. D'abord, le concepteur doit sélectionner les classes qui seront contenues dans ce schéma, ensuite le système procède à la génération automatique d'un schéma vue consistant et non redondant. La figure 6.(d) montre les classes sélectionnées et la figure 6.(e) montre le schéma vue crée.

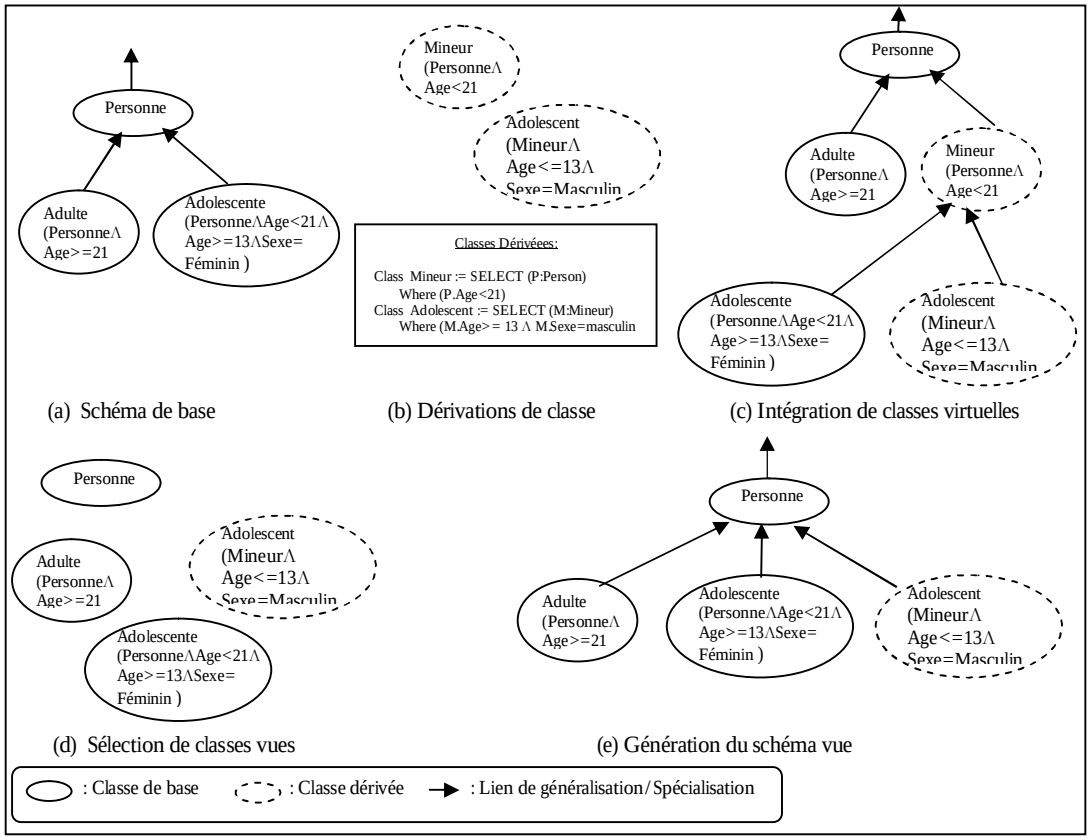


Figure 3. Définition d'un schéma de base, d'un schéma global et d'un schéma de vue

1.2.3. Modèle d'objets

L'implémentation du modèle d'objets dans MultiView repose sur la technique d'"*object-slicing*" [52] dont le principe est : une entité du domaine correspond à une hiérarchie *d'objets d'implémentation* (*implementation objets*, un pour chaque classe dont l'objet est instance) liée à *un objet conceptuel* (*conceptual object*, qui dénote l'entité elle-même).

Un objet peut donc être instance de plusieurs classes et avoir plusieurs types qu'il peut perdre ou acquérir dynamiquement (s'il devient instance d'une classe ou non). Les objets d'implémentation et l'objet conceptuel qui correspond à une même entité du domaine ont la même identité. Un objet d'implémentation correspondant à une classe C ne détient en propre que les propriétés localement définies dans C, et si un objet conceptuel est lié à cet objet d'implémentation, alors il doit également être lié à un objet d'implémentation pour chaque super-classe (directe ou indirecte) de C dans le schéma global. En d'autres termes, les objets d'implémentation liés à un objet conceptuel reflètent la hiérarchie des classes du schéma global dont l'objet conceptuel est instance, et de la même manière que la description du comportement et de la structure de cet objet conceptuel est fragmenté au niveau des classes, sa représentation est fragmentée au niveau de ses objets d'implémentation. La figure 4(a) représente un schéma composé de deux classes de base *Personne* et *Adolescente* et de deux classes virtuelles *Mineur* (dérivée par une requête de sélection à partir de la classe de base Personne) et *Sportive* (dérivée par enrichissement de la classe en ajoutant de nouvelles variables d'instance, type-sport et poids). La figure 4.(b) présente un objet conceptuel O1 et ses liens

d'implémentation qui le relient aux différentes classes (de base et virtuelles) dont il est instance.

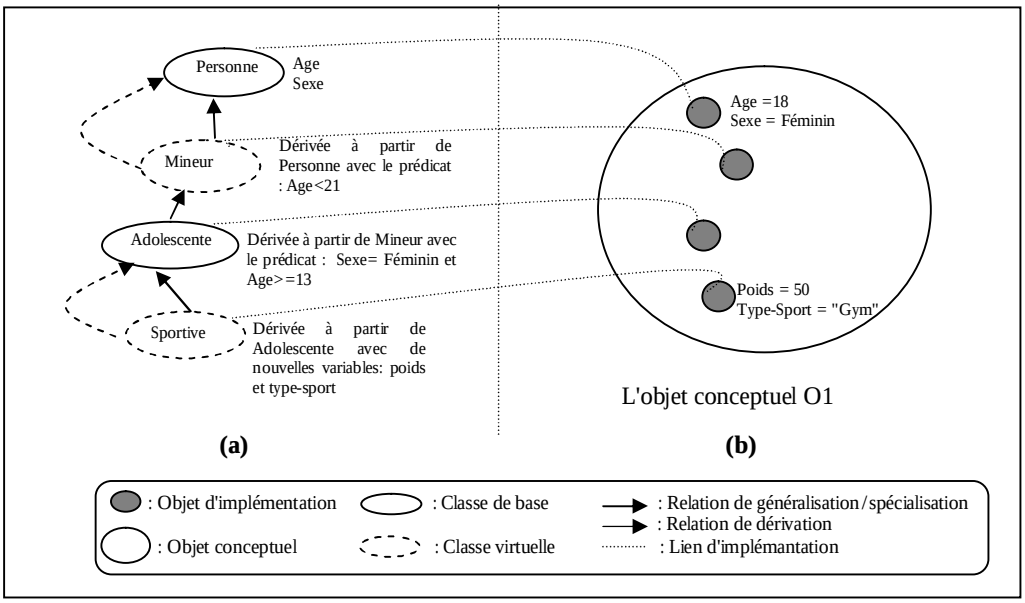


Figure 4. Exemple d'object-slicing

Un mécanisme d'héritage de propriétés semblables à la délégation permet d'hériter les propriétés entre objets d'implémentation liés à un même objet conceptuel : si un message envoyé à un objet d'implémentation dont la classe ne détient pas le sélecteur, celui-ci est alors recherché dans les super-classes et s'il est trouvé, la méthode correspondante est activée sur l'objet d'implémentation lié au même objet conceptuel.

1.2.4. Caractéristiques du modèle des vues dans MultiView

Le modèle des vues de MultiView se caractérise par :

- A l'instar du modèle des vues de COCOON, MultiView ne permet pas la création de nouveaux objets.
- Dans cette approche, les classes dérivées sont intégrées dans la hiérarchie des classes du schéma de base. La classification automatique des classes dérivées peut amener à la création de classes intermédiaires et à des restructurations de la hiérarchie des classes. Le coût de ces restructurations peut s'avérer important, bien que limité par certains résultats [53]. En effet, ce coût est contre balancé par les bénéfices de l'adoption de la technique d'object-slicing [52], qui a pour conséquence de matérialiser toutes les classes virtuelles de MultiView (et donc d'optimiser les temps nécessaires à l'interrogation de la base) sans pour autant introduire de redondance en ce qui concerne le stockage des informations.
- Avec la gestion des vues matérialisées dans MultiView, le problème de cohérence ne se pose plus, aussi l'unicité des informations directement accessibles et l'existence du schéma global offrent des possibilités intéressantes d'optimisation des opérations de mise à jour [513

1.2.5. Vues et points de vue

En comparaison avec la notion de point de vue qui est directement inhérente à la représentation de l'objet lui-même, les vues correspondent plus généralement à une façon de percevoir un système (une base de données) dans son ensemble.

Cette différence essentielle rend difficile la comparaison des deux approches et nous incite à conclure une divergence d'objectif et de niveau de conception. Cependant, si le concept de point de vue ne comporte que très peu de points communs avec les systèmes tels que COCOON, une certaine ressemblance apparaît avec le modèle MultiView et plus exactement avec la technique d'object-slicing utilisée dans la construction de son modèle d'objets. Dans les deux cas, i.e. les objets dans MultiView et les objets multi-points de vue, la représentation des entités du monde réel est fragmentée au travers de plusieurs représentations (objets d'implémentation dans MultiView) qui sont organisées conformément à une certaine structure (une hiérarchie des classes dont l'objet conceptuel est membre dans MultiView).

La représentation des vues au travers d'une représentation fragmentée des objets s'apparente à la représentation de fonctionnalités par des points de vue, sous l'hypothèse que chaque fragment de la représentation correspond à la représentation de cet objet dans le plan d'une fonctionnalité (le schéma global de MultiView assume en partie cette responsabilité). Il semble donc que les points de vue peuvent être utilisés dans ce contexte pour l'implémentation d'un modèle d'objets de système de gestion de bases de données.

2. Les rôles

Comme cela a été mentionné en introduction de ce chapitre, dans le modèle orienté objet, chaque objet est lié à une seule classe de manière exclusive et permanente. C'est cette classe qui détermine sa structure et son comportement et définit ainsi son rôle unique. Or, dans la réalité, un même objet peut acquérir ou perdre divers aspects structurels ou comportementaux au cours de son existence. L'évolution de tels objets doit donc obligatoirement être prise en compte par le modèle de données. Malheureusement, le modèle objet standard ne tient pas compte de cet aspect.

L'objectif des modèles objet à rôles [74] [44] [4] [70] [30] est d'étendre le modèle objet pour permettre à tout objet :

- de suivre son évolution en acquérant ou en perdant des rôles et par conséquent de changer de structure et de comportement pendant son cycle de vie,
- de posséder plusieurs rôles en même temps,
- d'avoir un comportement variable selon le rôle joué.

La gestion de la notion de rôle constitue un sujet de recherche actuel comme en témoigne les nombreuses publications tant en conception, modélisation des systèmes à objets, qu'en représentation des connaissances, langages de programmation et bases de données [30].

Le concept de rôle a été utilisé dans les bases de données classiques pour modéliser les différents rôles, facettes ou aspects d'une entité [74] [55] [70]. L'exemple classique d'une entité avec plusieurs rôles est celui d'une personne qui passe de l'état d'étudiant à

l'état de fonctionnaire puis à celui de directeur. Chaque rôle (étudiant, fonctionnaire ou directeur) correspond à une facette de l'entité réelle (la personne). L'entité est alors représentée par une structure commune qui est complétée par des informations relatives à ses différentes facettes. La notion de rôle offre donc une solution pour le support de la représentation multiple en permettant à chaque entité réelle d'être représentée par un ensemble d'objets ou instances appartenant à différentes classes qui correspondent aux rôles de l'entité.

Dans cette section, nous présentons, deux modèles de bases de données intégrant le concept de rôle; le modèle FIBONACCI et celui de S. Coulondre. Puis nous comparons le concept de rôle et celui de la représentation des objets selon des points de vue multiples.

2.1. FIBONACCI

FIBONACCI est un langage de programmation des bases de données à objets [4] [5], fortement typé qui intègre le mécanisme des rôles pour la modélisation du comportement dynamique et variable des objets au cours de leur vie. Il est basé sur les notions d'héritage, d'encapsulation, d'envoi de messages et de résolution tardive (late binding) et offre d'autres caractéristiques telle que la séparation explicite entre les types des objets et leurs implémentations d'une part et les objets et leurs rôles d'une autre part.

Un objet est défini comme un ensemble de rôles, ou agrégat de rôles. Il possède une identité propre qui persiste indépendamment de toute modification de sa structure ou de son comportement. Un objet ne peut être manipulé que par envoi de message au travers de l'un de ses rôles. De l'extérieur, il peut être considéré comme une boîte noire dotée d'autant d'interfaces (ensembles de méthodes) que de rôles dans l'objet. L'une des contributions originales du langage FIBONACCI est que le mécanisme d'interprétation de messages (recherche de la méthode à appliquer) qu'il utilise combine les deux mécanismes d'héritage et de délégation (voir § 2.3.2.3). Dans ce qui suit, nous présentons les concepts du modèle de données pour la définition, la création et la manipulation des objets avec rôles.

2.1.1. Classes objet-type et classes role-type

Dans FIBONACCI, on distingue deux types de classes : les classes *objet-type* et les classes *role-type*. Une classe *objet-type* est une classe abstraite, ne contenant pas de description particulière, et est définie par un ensemble de classes *role-type* qui lui sont associées. Les classes *role-type* qu'un objet peut acquérir sont organisées dans une hiérarchie de sous-typage avec héritage multiple qui admet pour racine la classe *objet-type*. Ce sont les classes *role-type* qui contiennent la description comportementale des rôles d'objets, c'est-à-dire les différentes méthodes pour les valeurs de leurs propriétés.

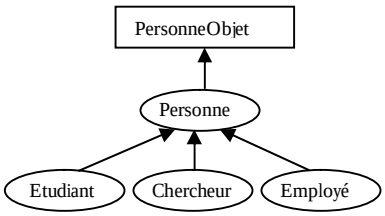
La figure 5 présente la définition d'un *objet-type* PersonneObjet et de l'ensemble de ses role-type.

```
Let PersonneObjet = NewObject;

Let Personne = Is-A PersonneObjet With
        Nom : String;
        Année-naiss: Int;
        Age : Int;
        QuiJe Suis: String;
        End;

Let Etudiant = Is-A Personne With
        Faculté : String;
        NumEtud: Int;
        QuiJe Suis: String;
        End;

Let Employé = Is-A Personne With
        Département : String;
        NumEmp: Int;
        QuiJe Suis: String;
        End;
……..
```



: Classec Objet-type : Classe role-type : Lien d'héritage

Figure 5. Définition d'un objet-type et de la hiérarchie des classes et role-type

A chaque role-type est associé un ensemble de méthodes qui constitue son interface. L'implémentation de ces méthodes est spécifiée séparément dans des constructeurs d'objets. Cependant, dans FIBONACCI, on offre deux méthodes pour construire des objets avec rôles : soit d'une manière individuelle ou encore en utilisant des constructeurs génériques d'objets. Par la première approche, des objets ayant les mêmes rôles peuvent avoir des implémentations différentes de leurs méthodes. Dans ce qui suit nous présentons la deuxième approche que nous jugeons la plus générale.

2.1.2. Constructeurs d'objets

Un constructeur d'objets est une fonction définie dans un role-type qui permet la création des objets avec un certain rôle. Les objets crées partagent la même structure et l'interface spécifiées dans la classe role-type et la même implémentation des méthodes définies dans le constructeur de celle-ci. Un constructeur d'un role-type (excepté le role-type ne possédant pas de super role-type) peut être défini à partir de *nil* ou par extension d'un constructeur générique défini dans son(ses) super role-type. L'expression **fun** (<arguments>) : <type> **is** <expression> définit la fonction de création d'un objet dans un rôle donné. La figure 6 montre la définition du constructeur d'objets du role-type Personne, qui ne possède pas un super role-type et est donc défini à partir de nil, et du constructeur du role-type Etudiant défini par extension du role-type Personne.

```
/* Création du constructeur du role-type Personne */
Let  CreatePersonne = fun (N : String ; Année : Int ; Adr : String) : Personne  is
    role  Personne
    private
        let  Nom = N;
        let  Année-naiss = Année;
        let  Adresse = Adr;
    methods
        MonNom = Nom
        MonAdresse = Adresse
        MonAge = currentYear() – Année-naiss;
        QuiJeSuis = "Mon nom est :  ", Nom, "Mon age est : ", self . age ;
End;

/* Création du constructeur du role-type Etudiant */
Let  CreateEtudiant = fun (N, Adr, Fac : String ; Année, Num : Int) : Etudiant
is ext  CreatePersonne (N : String ; Année : Int ; Adr : String) to Etudiant
    private
        let  Faculté = Fac;
        let  NumEtud = Num;
    methods
        QuiJeSuis = "Mon nom est : ", Nom, "Mon age est : ", self . age , "Ma
        MaFaculté = ", Fac, "Mon numéro d'étudiant est : ", Num;
End;
```

Figure 6. Définition de constructeurs d'objets

La création d'un objet instance du role-type Personne se fait par l'expression **Let** p = createPersonne ("Mohamed", 1970, "5, Rue Didouche Mourad"). A l'objet P peut être attaché le rôle d'étudiant en lui appliquant le constructeur createEtudiant. Ainsi, l'expression Let P-Etud = createEtudiant (P; "Science de l'ingénieur", 125468) permet d'étendre l'objet P par un nouveau rôle. Dans la figure 7, nous présentons, la structure de l'objet P et de son extension P-Etud.

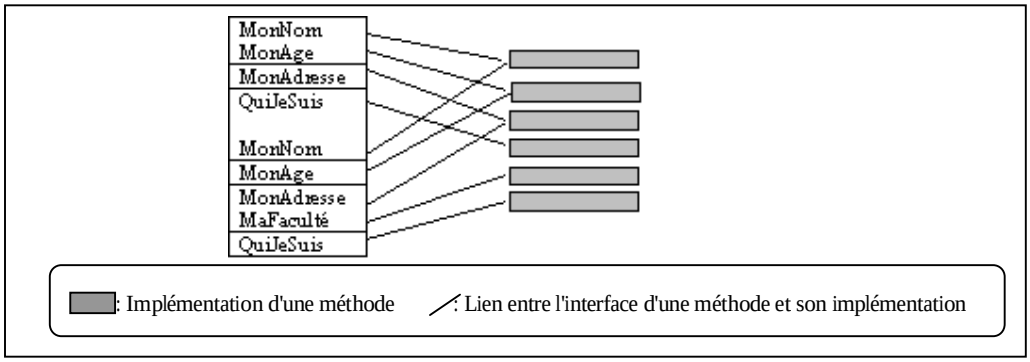


Figure 7. La structure de l'objet P et de son extension P-Etud.

Le constructeur défini par extension permet de modéliser l'aspect évolutif du comportement des entités et permet à un objet d'être étendu dynamiquement par de nouveaux sous-roles. Et parce que l'objet est une structure qui englobe tous ses rôles, il est possible de demander à un objet quels rôles il détient à tout moment (*role inspection*).

2.1.3. Traitement des messages

Les différentes interfaces des rôles sont connectées à un mécanisme interne à l'objet appelé le *dispatcher* qui redistribue les messages envoyés à l'objet aux différentes implémentations des méthodes dans les différents rôles de celui-ci. Ce mécanisme repose sur les principes suivants :

- Le comportement le plus spécialisé prévaut (*late binding*), c'est-à-dire que l'évaluation d'un message dont le sélecteur est détenu par deux rôles dont l'un est sous-type de l'autre conduit à l'évaluation de la méthode telle qu'elle est définie dans le contexte du premier rôle (le plus spécifique). On parle alors de partage par **délégation**. Par exemple, le message QuiJeSuis envoyé à P **as** Personne provoque l'activation de la méthode QuiJeSuis du role-type Etudiant. Par contre, le message Nom envoyé à l'objet P-Etud provoque l'activation par **héritage** de la méthode définie dans le role-type Personne. La méthode Nom n'étant pas redéfinie dans le role-type Etudiant.
- Les comportements deviennent plus spécialisés dans le temps, lorsque le sélecteur du message est défini pour plusieurs rôles qui sont incomparables, l'ordre d'ajout des rôles est pris en compte et un rôle est, en l'absence de toute autre information, considéré plus spécifique que les rôles ayant été antérieurement ajoutés à l'objet. Par exemple, la méthode QuiJeSuis dans la classe role-type Employé est choisie pour activation par le mécanisme d'héritage si elle a été définie après celle de la classe role-type Etudiant.

2.1.4. Caractéristiques des rôles dans FIBONACCI

Le mécanisme de rôles dans le langage FIBONACCI se caractérise par :

- La représentation multiple est centralisée au niveau des objets. L'état d'un objet est étendu en ajoutant un rôle, ce qui ne permet pas l'accès à ce dernier indépendamment de l'objet. Cependant, le mécanisme de résolution des méthodes évite l'interférence entre les différents rôles.
- Les objets ne possèdent pas une description de base qui caractérise les propriétés intrinsèques des objets. Cependant, l'organisation des rôles en une hiérarchie permet le partage des propriétés entre eux.
- Le mécanisme d'attribution des rôles aux objets est très flexible. Un rôle associé à un objet peut être crée à partir d'un constructeur d'objets ou d'une manière individuelle en personnalisant l'implémentation des méthodes de chaque objet. Aussi, un constructeur d'objets peut être défini à partir de nil en reprenant toutes les propriétés de la hiérarchie des rôles ou encore par extension d'un constructeur défini dans un rôle-type générique (un super role-type).
- La résolution des méthodes se base sur les deux mécanismes d'héritage et de délégation.
- Les rôles sont gérés dynamiquement par ajout et suppression. La suppression dans un objet d'un rôle entraîne la suppression de tous les rôles qui lui sont plus spécifiques. Aussi, un objet est capable d'intercepter toute tentative d'accès à un rôle ayant été supprimé et de provoquer une erreur en conséquence.

2.2. Le modèle de S. Coulondre

Le modèle de S. Coulondre [30] est une extension du modèle objet ODMG avec le mécanisme de rôle. L'idée de base est tirée de la constatation suivante : puisque le concept de classe est utilisé pour modéliser la structure et le comportement des objets, il en résulte que celui-ci peut être étendu afin de supporter les rôles des objets qui représentent la modélisation de l'évolution de la structure et du comportement de ceux-ci au travers de leur vie. En conséquence, d'autres extensions sont apportées au système de typage, au mécanisme d'héritage, à la résolution tardive des méthodes ainsi qu'aux langages de définition et de manipulation des requêtes. Le modèle intègre également un mécanisme limité des vues. Dans ce qui suit, nous présentons les concepts de base du modèle qui sont inhérents au mécanisme de rôle. Nous ne nous intéressons pas ici au mécanisme des vues ni aux langages.

2.2.1. Classes et rôles de classes

Le modèle distingue entre deux types de classes : les *classes* qui permettent l'instanciation et la création des objets et les *classe-roles* qui modélisent les rôles dans une classe. Les classes sont organisées en une hiérarchie d'héritage. Une classe ne possède pas de type ni de méthodes. Elle est définie par un ensemble de classe-roles organisées, à leur tour, en une hiérarchie par le lien *extension-de*. La figure 8 présente un exemple d'une hiérarchie de classes modélisant les entités du monde réel. La hiérarchie est composée des classes suivantes : *entité, animal, plante, matière et humain*. Chaque classe est un conteneur de classe-roles. La classe *entité*, par exemple, est composée des classe-roles : *commun, physique et chimique*.

Une classe-role dans une classe est définie par un critère, un type et un ensemble de méthodes. Un critère est une formule de la logique du premier ordre qui reflète une sémantique librement exprimée par l'utilisateur. Dans la figure 8, la classe Animal est constituée des classe-roles suivantes (exprimées par des noms pour la clarté de la présentation) : *commun, physique, chimique, biologique, prédateur et proie*. Ces classe-roles peuvent être définies, respectivement, par les critères suivants : vrai, état(*physique*), état(*chimique*), contexte(*biologique*), contexte(*biologique*) $\wedge$ état(*prédateur*), et contexte(*biologique*) $\wedge$ état(*proie*).

La sémantique des critères engendre, par implication, un ordre partiel entre les classe-roles. Par exemple, contexte(*biologique*) $\wedge$ état(*prédateur*) implique contexte(*biologique*).

Le lien sous-classe-de qui relie les classes n'est pas régi par une contrainte de sous-typage (une classe ne possède pas de type), mais par des contraintes sur leur hiérarchie de classe-roles. Une sous-classe hérite la hiérarchie des classe-roles de sa super-classe qu'elle peut spécialiser en ajoutant de nouvelles classe-roles, en affinant le type ou encore en ajoutant de nouvelles propriétés aux classe-roles existantes.

Un lien additif *sous-rôle-de* est engendré par la relation *sous-classe-de* qui permet de relier implicitement deux mêmes classe-rôles définies dans deux classes différentes dont l'une est sous-classe de l'autre. Dans la figure 8, la classe-rôle *biologique*, définie dans la classe *animal*, est redéfinie dans la sous-classe *humain*.

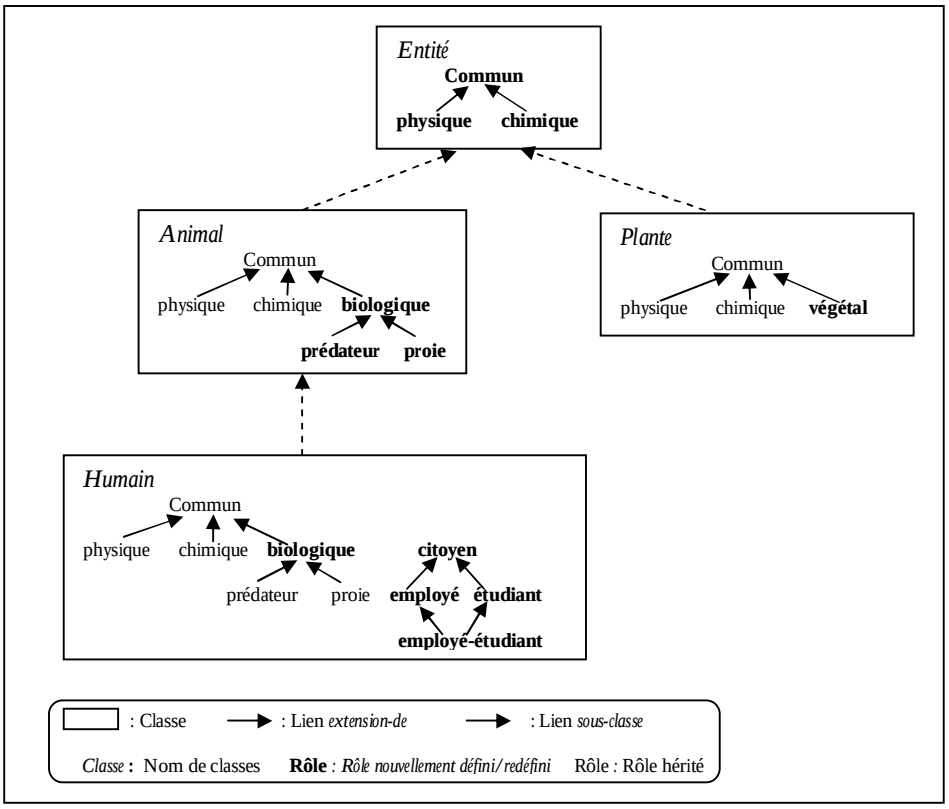


Figure 8. Les hiérarchies de classes et de classe-rôles

2.2.2. Objets et rôles d'objets

Un objet est une instance d'une seule classe et peut avoir un ou plusieurs rôles définis dans cette classe. Un identificateur unique est affecté pour chaque objet. L'ensemble des rôles d'un objet est un sous-ensemble de la hiérarchie des classe-rôles de sa classe d'instanciation. A chaque rôle d'un objet est affecté une valeur conforme au type de la classe-rôle associée. La figure 9 montre la structure de l'objet Mohamed d'OID 120. Cet objet joue cinq rôles : *commun, physique, chimique, citoyen et étudiant.*

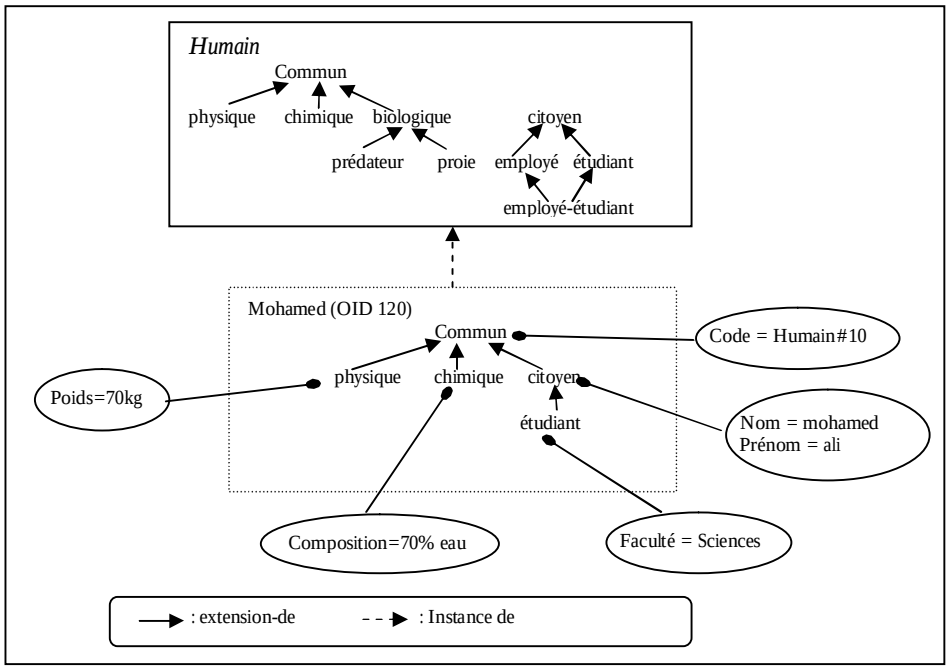


Figure 9. Présentation d'une instance de classe avec rôles

2.2.3. Envoi et interprétation des messages

Tout message n'est pas directement envoyé à un objet mais à un de ses rôles. Un message est composé de l'identifiant de l'objet, du critère qui définit son rôle et de la méthode à exécuter. Le processus de résolution dynamique d'un message est le suivant :

Soit C la classe d'appartenance de l'objet (o, w), w étant le critère du rôle de l'objet o, et m la méthode à exécuter :

- Partant de la classe-rôle w dans la classe C, on remonte dans la hiérarchie des super-rôles de w induit par le lien *sous-rôle-de* jusqu'à la racine. La méthode est invoquée à la première rencontre,
- Si la méthode n'est retrouvée, la recherche redémarre à partir du super-rôle de w dans la hiérarchie des classe-rôles de C, c'est-à-dire en suivant le lien *est-extension-de*.

Par exemple, soit le message (Mohamed, (contexte(*biologique*))→ afficher( ) envoyé à l'objet Mohamed jouant le rôle spécifié par le critère contexte(*biologique*). La figure 10 illustre le processus de recherche de la méthode afficher( ). La recherche s'effectue dans la classe-rôle *biologique* de la classe *humain*, puis dans la classe-rôle *biologique* da la classe *animal*. Celle-ci est la racine de la hiérarchie des super-rôles étant donné que la classe *commun* ne possède pas le rôle *biologique*. Si la méthode n'est pas retrouvée, le même processus de recherche redémarre à partir du super-rôle de *biologique* dans la classe *humain*, i.e., la classe-rôle *commun* et ainsi de suite jusqu'à atteindre la racine de la hiérarchie des classe-roles de *humain*.

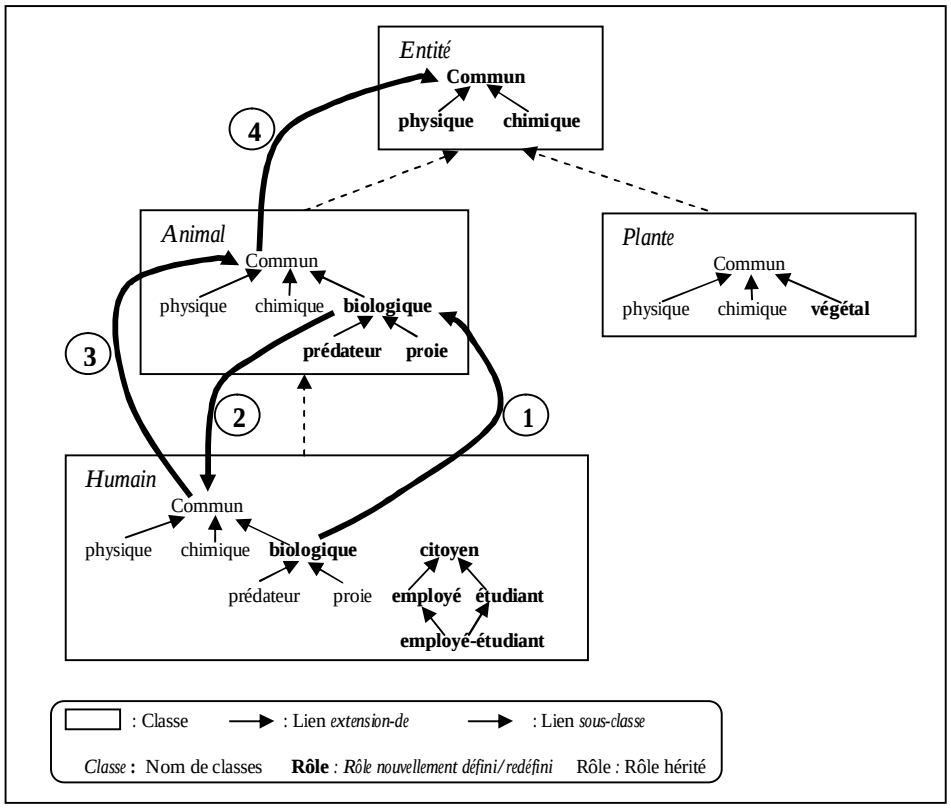


Figure 10. Le processus de résolution des méthodes

2.2.4. Caractéristiques des rôles dans le modèle de Coulondre

Le modèle de Coulondre se caractérise par :

- Il repose sur une idée originale dont l'emphase est mise sur la définition d'une classe par une hiérarchie de classe-rôles. Une hiérarchie de classe-rôles est partagée par une classe et ses sous-classes.
- La représentation multiple est centralisée au niveau de l'objet. Un objet est instance d'une classe et peut être attaché à un ou plusieurs de ses rôles.
- Les rôles permettent un partage de propriétés par héritage qui peut être multiple.

3. Vues, rôles et points de vue

L'intégration de la notion de point de vue dans les bases de données à objets permet d'intégrer l'agent et sa vision par rapport au système dans une étape tardive. En général le point de vue apparaît à l'exploitation des données à l'instar des vues qui sont exploitées par les langages d'interrogation comme des fonctions de sélection sur les données ou encore les rôles qui représentent l'évolution ou la dynamique des objets tout au long de leur durée de vie. L'essence des points de vue est la prise en compte des agents dès les premières étapes d'analyse et de conception des données. Dans la figure 11, nous schématisons l'intégration des trois concepts dans un schéma de base de données. Notre approche, présentée en figure 11.(c), est centrée sur l'interprétation de la relation "agent-monde" qui suppose que les différents points de vue sur un même univers de discours sont des visions partielles mais complémentaires (voir § 1.2.1.). Elle se base essentiellement sur une conception décentralisée d'un schéma de base de données selon des points de vue multiples. Chaque concepteur et suivant sa vision d'un même univers de discours, ayant une représentation de base (schéma de base), construit un schéma partiel que nous appelons "schéma-point de vue". Les différents schémas points de vue ne sont pas isolés mais s'échangent des informations et leur union produit une représentation cohérente du monde.

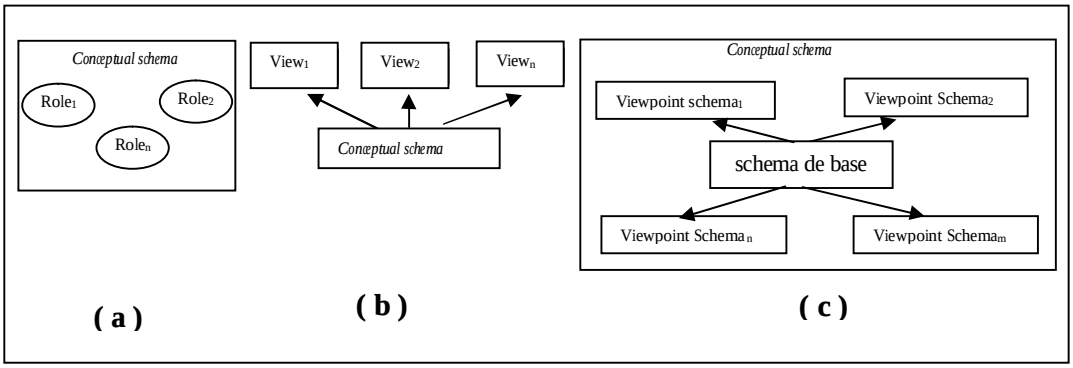


Figure 11. Intégration des rôles, vues et points de vue dans un schéma de base de données

Les trois mécanismes de vues, rôles et points de vue tentent tous d'étendre les concepts du modèle objets pour la prise en compte de certains aspects tels que la représentation multiple et l'évolution des objets. Dans ce qui suit, nous nous proposons de les comparer en fonction de quelques critères :

1. Objectifs

- Une vue est un schéma externe défini dans le but d'adapter une structure existante à de nouveaux besoins,
- Un rôle est défini pour représenter l'évolution dynamique des objets, c'est-à-dire ses comportements variables dans le temps,
- Un point de vue est défini afin de permettre aux objets d'un même univers de discours d'avoir simultanément plusieurs représentations partielles mais complémentaires.

2. Niveau de description

- Une vue est définie au niveau du système et dénote un point de vue sur le domaine dans son ensemble pour, éventuellement, ne considérer que la description d'une partie de celui-ci. Une vue concerne plutôt des aspects externes à l'objet en liaison avec les utilisateurs.
- Un rôle, tout comme un point de vue, est lié directement à la description de l'objet. La description complète de ce dernier se construit à partir des rôles ou points de vue qui lui sont inhérents.

3. L'impact sur la description d'un objet

- Une vue repose sur la description contenant la presque totalité des informations initiales sur un objet (à quelques attributs dérivés près). Elle n'apporte pas de richesse à la description de base de l'objet.
- Un rôle, tout comme un point de vue, confère à l'objet de nouvelles propriétés non dérivées ou calculées à partir d'autres déjà existantes.

4. Communication et partage des données

- Les vues sont introduites afin d'assurer l'indépendance des données. De ce fait, deux vues, dérivées d'un même schéma racine, ne peuvent pas partager des propriétés encore moins communiquer entre elles. De la même manière, deux rôles associés à un objet se partagent uniquement les propriétés intrinsèques,
- Vu le principe de complémentarité qui existe entre les points de vue, le partage des données et la communication sont donc des caractéristiques essentielles à gérer.

5. Traitement de la cohérence

- La cohérence entre deux vues indépendantes n'est pas traitée. Cependant l'étude de la cohérence se restreint juste à la cohérence de la définition d'une vue par rapport au schéma racine ou au schéma vue dont elle est dérivée,
- L'étude de la cohérence dans les rôles consiste à assurer la cohérence du comportement des objets à travers leur évolution,
- La cohérence dans les points de vue se situe à deux niveaux : un objet doit satisfaire les contraintes locales dans chaque représentation partielle puis les contraintes globales exprimées sur l'ensemble des représentations partielles.

4. Conclusion

Dans ce chapitre, nous avons présenté l'utilisation de l'approche par point de vue dans le domaine des bases de données à objets. Nous avons vu les mécanismes, en particulier la notion de vue et celle de rôle, permettant une représentation multiple qui est une caractéristique de base pour un modèle de données à objet. Une comparaison entre ces approches a été effectuée et qui nous a permis de dégager les forces et les limites de chacune. Ensuite, nous avons évoqué succinctement les apports de notre approche que nous présentons dans le prochain chapitre.

Chapitre 3

*Où l'on présentera l'approche de MVDB (Multi-Viewpoint-DataBase), un modèle de données qui permet la représentation **multiple**, **évolutive** et **distribuée** des données dans le cadre des bases de données orientées objets. Cette représentation confère à un même univers de discours plusieurs représentations partielles et complémentaires. Celles-ci reposent sur une description de base des entités et l'étendent selon différentes perspectives ou points de vue. La représentation multiple offre une échappatoire à la contrainte d'instanciation unique de l'approche objet, la représentation évolutive remet en question la représentation figée des objets et la représentation distribuée répond aux exigences des applications actuelles de développement réparti et décentralisé des bases de données.*

Nous adoptons le modèle objet comme le modèle commun pour les différents schémas de la base de données. Ce choix se justifie par trois motivations principales. Tout d'abord, on reconnaît à l'approche objet une puissance d'expression et de structuration des données qui coïncident avec les caractéristiques de modélisation des objets multi-points de vue, telle que le mécanisme de multi-instantiation qui permet à un objet d'être instance de plus d'une classe. Deuxièmement, la technologie objet a été utilisée dans les systèmes multidatabase à une granularité fine. Troisièmement, l'application des concepts orientés objet dans les architectures des systèmes confère à ceux-ci un modèle naturel d'autonomie et de distribution.

Notre principale problématique est la construction de schémas de bases de données selon différents points de vue appelés "schémas multi-points de vue". Un schéma multi-points de vue est composé d'un schéma de base "le référentiel" qui est une partie partagée par tous les concepteurs de la base de données et d'un ensemble de schémas appelés "schémas points de vue" qui représentent une extension du référentiel. Chacun donne lieu à une structuration particulière et spécifique des entités selon un point de vue particulier. L'aspect pluridisciplinaire d'une base de données se reflète donc dans l'ensemble des points de vue. Les points de vue peuvent être liés par des relations qui reflètent l'aspect interdisciplinaire d'une base de données et s'avèrent utiles pour la complémentarité de la représentation multiple des données.

MVDB offre des mécanismes pour spécialiser le schéma du référentiel, spécifier les dépendances entre les différents schémas points de vue, assurer la cohérence du modèle en cas de mise à jour, etc. La sémantique du modèle est décrite moyennant une extension du modèle objet standard. Cependant, nous nous situons dans le cadre d'un modèle d'objets serveurs d'information dont l'encapsulation est peu justifiée, leur structure est directement accessible par les clients de la base de données et leur cohérence est garantie par des contraintes d'intégrité. Par conséquent, l'accent est mis exclusivement sur l'aspect "données" de ces objets sans prendre en compte les aspects relatifs aux traitements[1]. Nous adoptons le modèle "objet" de l'ODMG au travers du langage ODL qui nous sert comme support pour les descriptions des concepts de MVDB à un niveau logique. Le choix de l'ODMG est justifié principalement par le fait qu'il propose un modèle abstrait pour la définition des schémas de bases de données objet en utilisant le langage "ODL". En effet, une spécification d'un schéma objet

[1] *Il est cependant possible d'introduire des accesseurs.*

décrite en ODL est indépendante d'un SGBD précis. Lors de l'implémentation, elle peut être traduite dans le langage cible.

Le reste du chapitre est structuré de la manière suivante. La section 1 présente la méthodologie de notre modèle en décrivant d'une manière informelle ses principes et ses concepts de base intégrant le mécanisme de point de vue. Une description formelle de ces concepts sera présentée dans la deuxième section.

1. La méthodologie de MVDB

La méthodologie du modèle que nous proposons repose sur une critique principale du modèle objets à base de classes standard : les concepts de base de ce dernier permettent de modéliser une représentation unique d'un univers de discours qui ne tient pas compte de la diversité des visions qu'on peut avoir sur les objets du monde réel. Dans ce modèle, un schéma de base de données est composé d'une hiérarchie de classes qui confère aux objets une description unique et figée. La proposition apportée par notre modèle consiste à doter les concepts du modèle objet standard d'une capacité de représentation multiple et évolutive en intégrant la notion de point de vue. Dans cette section, nous présentons tout d'abord les principes de MVDB puis nous définissons d'une façon informelle les différents concepts introduits. En guise d'illustration, nous nous appuyions sur un exemple simple de modélisation d'une base de données et qui concerne une application de Gestion d'une Entreprise de Fabrication et de Distribution de Produits (GEF-DP). Cet exemple est décrit en détails en annexe A.

1.1. Principes de base

Les points de vue dans MVDB suivent l'hypothèse d'un monde unique vu de façons complémentaires par différents observateurs (voir §1.2.1. du chapitre 1). Parmi les objectifs essentiels de notre approche, nous citons :

- Offrir une vision de développement décentralisé d'un schéma de base de données.
- Favoriser une représentation des données qui tient compte de plusieurs points de vue. Ceci facilite le travail en parallèle de plusieurs équipes de conception.
- Favoriser un échange d'informations entre les descriptions partielles du schéma qui ne sont pas isolées.
- Assurer la cohérence du schéma global décrit par plusieurs points de vue,
- Tirer profit de la représentation multiple des données pour permettre leur évolution.

Actuellement ces objectifs sont très souhaitables notamment dans les grandes applications complexes du milieu industriel. En effet, les entreprises sont normalement déjà distribuées, au moins logiquement (en divisions, départements, groupes de travail, etc.) d'où l'on peut déduire que les données sont également déjà distribuées, car chaque unité organisationnelle dans l'entreprise doit nécessairement gérer les données pertinentes pour son fonctionnement et devrait pouvoir accéder aux données éloignées des autres unités si nécessaire. Les données dans les différentes unités se complètent et fournissent une base de données globale qui permet de gérer l'entreprise. Une exploitation à bon escient et cohérente de cette base globale est ainsi recommandée.

Nous illustrons l'approche multi-point de vue par l'exemple d'une entreprise industrielle de fabrication et de distribution de produits. On s'intéresse tout particulièrement aux activités commerciale, comptable, de fabrication et de stock. Considérons les points de vue suivants : le point de vue "*commercial*", le point de vue "*comptable*", le point de vue "*fabrication*" et le point de vue "*stock*". Une représentation de base ou minimale des entités du domaine est constituée de l'ensemble des informations intrinsèques et communes à tous les points de vue. Ici, elle est constituée des informations élémentaires sur les produits, les clients, les fournisseurs, les bons de commande, les bons de livraisons et les dépôts. En revanche, chaque point de vue renferme seulement les informations qui lui sont pertinentes. Le point de vue *stock*, par exemple, étend la description de base par de nouvelles données relatives au point de vue telles que l'adresse et le téléphone de livraison pour le client, le responsable de livraison pour le fournisseur, le coût d'achat, le coût de stockage, et la quantité stockée pour le produit, etc. La figure 1 présente une modélisation multi-points de vue de base de données.

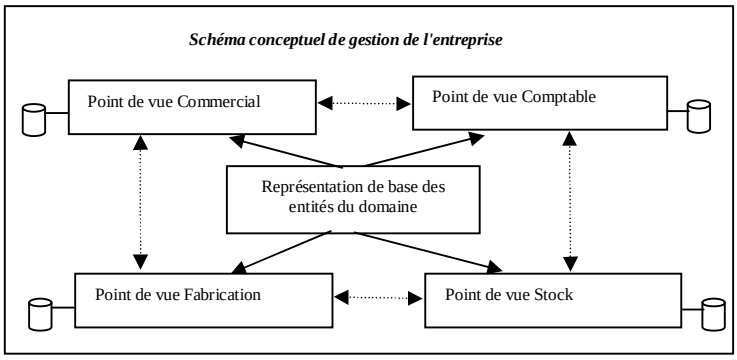


Figure 1: Exemple de modélisation

MVDB adopte les principes de modélisation suivants:

- Le schéma d'une base de données est une description multiple d'un même univers de discours selon différents points de vue. Un point de vue est une abstraction d'une certaine perception (vision) des objets du monde. Il ne correspond pas à une classe mais c'est une spécification d'une hiérarchie de classes appelée schéma Point de Vue (*schéma-PV*). Le schéma d'une base de données est donc composé d'un ensemble de schémas point de vue et est appelé schéma *multi-point de vue*. Chaque schéma-Point de vue décrit un aspect de la description des données et est détenu par un système de base de données indépendant.
- La construction des schémas-PV est basée sur un schéma de base appelé le *référentiel*. Ce dernier détient les propriétés de base des entités de l'univers de discours et qui sont partagées par tous les schémas-PV. Chaque schéma-PV étend la description de tout l'ensemble ou d'un sous ensemble des entités du référentiel selon un point de vue donné. Cette extension est régit par une relation d'extension et des règles de bonne construction.
- Les différentes représentations partielles peuvent partager les informations via une relation de dépendance. Celle-ci doit assurer la protection des informations échangées.

- Les objets qui sont les instances dans les différents points de vue contiennent les données du domaine d'étude (ou du système). Un objet peut être représentatif dans plusieurs points de vue. Le lien entre les différentes instances d'un objet est réalisé à travers la notion d'identité d'objet. Un objet est identifié d'une manière unique dans tous les schémas-PV. L'univers complet d'un système est l'union des différents schemas-PV.
- Les objets de la base de données ont une description de base dans le référentiel et une ou plusieurs descriptions selon les différents points de vue, on distingue un objet multi-point de vue et un objet point de vue respectivement.

1.2. Présentation des concepts de base

La notion clé sur laquelle repose les concepts de notre modèle est le "point de vue". Ainsi les entités standard de modélisation d'une base de données orientée objet: schéma de base de données, objet et base d'objets sont étendus par les concepts de *schéma multi-points de vue* de base de données, *d'objet multi-points* de vue et de *base d'objets multi-points de vue*, respectivement. Chacun de ces concepts renferment deux types d'informations:

- des informations intrinsèques qui représentent la description de base sur laquelle s'appuient les points de vue et,
- des informations spécifiques relatives aux descriptions selon les différents points de vue.

A ces concepts s'ajoutent ceux de *schéma point de vue, objet point de vue* et *base d'objets point de vue* qui représentent les entités de modélisation au niveau d'un point de vue. Cependant chaque entité de modélisation point de vue est liée à l'entité multipoints de vue dont elle représente une description partielle selon un point de vue donné par une relation spéciale notée Vp-Extension (pour Viewpoint-Extension). Pour chaque entité de modélisation, la relation impose des contraintes de bonne construction afin d'assurer la cohérence de la représentation multiple des données. (voir §2.2.).

Pour la clarté de l'exposé, nous accompagnons la définition de chacun des concepts de notre modèle d'une représentation schématique portant sur notre application.

1.2.1. Les schémas

L'approche de MVDB prône qu'un schéma multi-points de vue est composé d'un schéma référentiel et d'un ensemble de schémas points de vue. La notion de référentiel commun constitué des entités de base du domaine est centrale dans notre démarche. En effet, tout schéma point de vue est une extension descriptive du schéma du référentiel selon un point de vue donné. Les schémas dans MVDB sont des hiérarchies de classes ayant une structure d'arbres. Cependant l'héritage multiple n'est pas géré dans notre modèle.

1.2.2.1. Schéma référentiel

Le référentiel est constitué d'un ensemble de classes liées par la relation classique de généralisation/spécialisation. Chacune de ses classes détient la description de base commune aux différents points de vue. Ces derniers partagent cette description commune au travers des entités du domaine. MVDB n'impose pas de contraintes sur le

contenu de ce référentiel. Celui-ci peut fournir une représentation plus ou moins riche du domaine, en fonction du niveau de partage requis par les points de vue.

1.2.2.2. Schéma point de vue

Un schéma point de vue représente une extension descriptive des entités du référentiel selon un point de vue donné. Il a la structure d'arbre et est obtenu en sélectionnant la partie du référentiel (i.e. les entités du domaine) à décrire puis en procédant à son extension par les caractéristiques propres au point de vue. De ce fait, chaque entité du référentiel participe à l'ensemble (ou à un sous-ensemble) des points de vue. Dualement, les points de vue sont pertinents pour l'ensemble (ou un sous-ensemble) des entités du référentiel.

La figure 2 présente le référentiel de base de notre application et son extension selon le point de vue "*Service Après Vente*" (SAV) et le point de vue "*Comptable*". Cette extension est régit par la relation Vp-Extension dont les caractéristiques seront présentées formellement dans la section 2. Dans un souci de clarté, seule l'extension de l'entité Produit est portée sur la figure 2, une spécification complète, en langage ODL, du schéma référentiel des différents schémas points de vue est donnée en Annexe A.

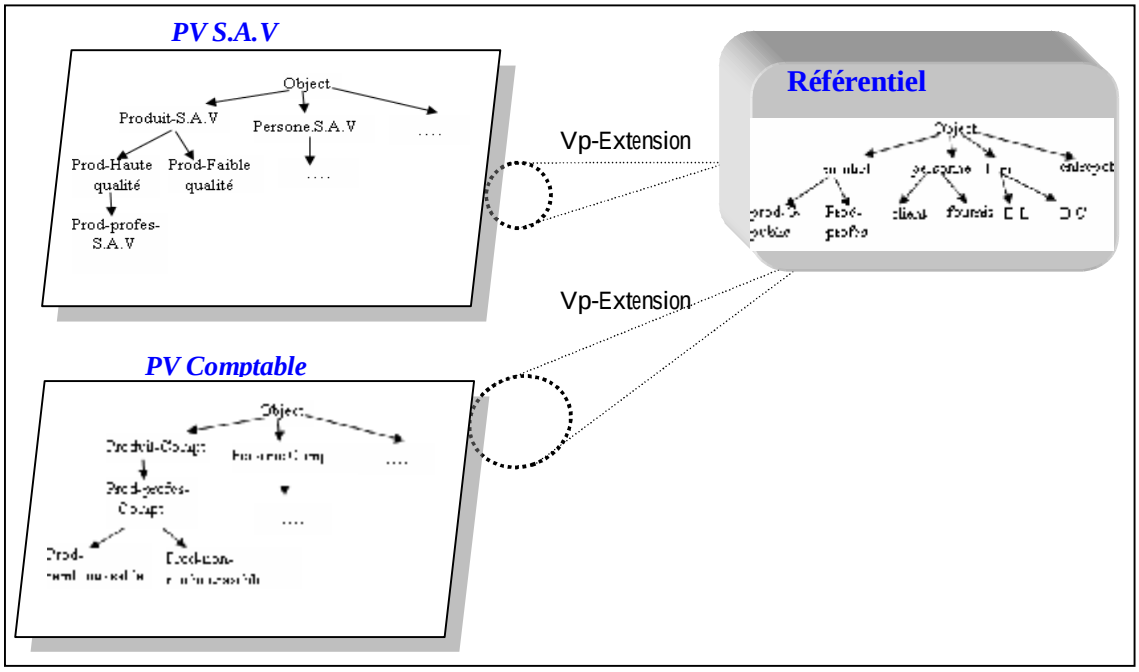


Figure 2. Exemple de schéma multi-points de vue

1.2.2.3. La dépendance entre les points de vue

Comme nous l'avons présenté ci-dessus, les différents points de vue d'un schéma de base de données produisent des arbres de classes différents. Cependant, ces arbres ne sont pas complètement indépendants les uns des autres. Une dépendance inhérente au modèle MVDB consiste en la représentation du même ensemble potentiel d'entités du référentiel dans les différents schémas points de vue. Ce qui engendre un besoin de partage des informations entre les points de vue. Deux formes de partage se distinguent:

• les informations communes propres aux entités et qui existent indépendamment des points de vue. Elles sont implicitement partagées par tous ces derniers. Par exemple, la désignation des produits est partagée et exploitée par les clients de tous les points de vue,

• les points de vue se partagent les informations qui leurs sont pertinentes. Ces informations sont issues de l'unique point de vue "fournisseur" qui détermine leur existence. Elles sont exploitées par un ensemble de points de vue "consommateurs" qui sont autant de clients internes du point de vue fournisseur. Ce partage doit être spécifié explicitement pour chaque point de vue consommateur. Par exemple, la description de la classe client dans le point de vue "Fabrication" s'intéresse au nombre d'unités vendues de chaque produit, information pertinente fournie et gérée par le point de vue "commercial" ainsi qu'au coût de stockage de chaque produit, information pertinente fournie par le point de vue "Stock". Ces dépendances qui sont des caractéristiques du système modélisé, doivent être explicitées durant la modélisation, pour cela nous utilisons une relation de dépendance notée Vp-Dependency qui permet de définir des passerelles entre les classes des différents points de vue et qui sera détaillée au prochain chapitre.

1.2.2. Les objets

Les objets sont les instances des classes dans les différents schémas d'une base de données multi-points de vue : le schéma référentiel et les schémas points de vue. L'hypothèse de l'exclusivité des classes soeurs d'un arbre entraîne la **mono-instantiation** au niveau de chaque schéma. La représentation multiple dans MVDB distingue deux types d'objets : les objets multi-points de vue et les objets points de vue.

1.2.2.1. Objet multi-points de vue

Un objet multi-points de vue est une représentation multiple d'une entité réelle. Il peut avoir autant de lien d'instanciation qu'il y'a de points de vue permettant de le représenter. MVDB autorise donc la **multi-instanciation** au niveau d'une base de données multi-points de vue. De ce fait, un objet multi-points de vue possède plusieurs états :

- un état de base constitué de l'ensemble des valeurs des propriétés intrinsèques définies au niveau du référentiel,
- des états spécifiques qui décrivent l'objet selon les différents points de vue. Ces états sont répartis dans des objets à part entière appelés des objets points de vue.

1.2.2.2. Objet point de vue

Un objet point de vue est une représentation partielle d'un objet multi-points de vue selon un point de vue donné. Chaque objet point de vue est relié à son objet multi-points de vue par la relation Vp-Extension qui traduit la délégation, un mécanisme absent dans le modèle objet classique à base de classes et que nous adaptons pour le partage des propriétés entre objets (Cf. §1.2. du chapitre 4).

La figure 3 schématise les liens d'instanciation et la relation Vp-Extension au niveau d'un objet multi-points de vue. L'objet P1 est un produit professionnel (instance de la classe "Prod-profes" du référentiel). P1 est représenté dans les deux points de vue *S.A.V*

et *Comptable* par les objets points de vue P11 et P12. P1 est un produit professionnel de haute qualité du point de vue S.A.V et un produit non remboursable du point de vue Comptable.

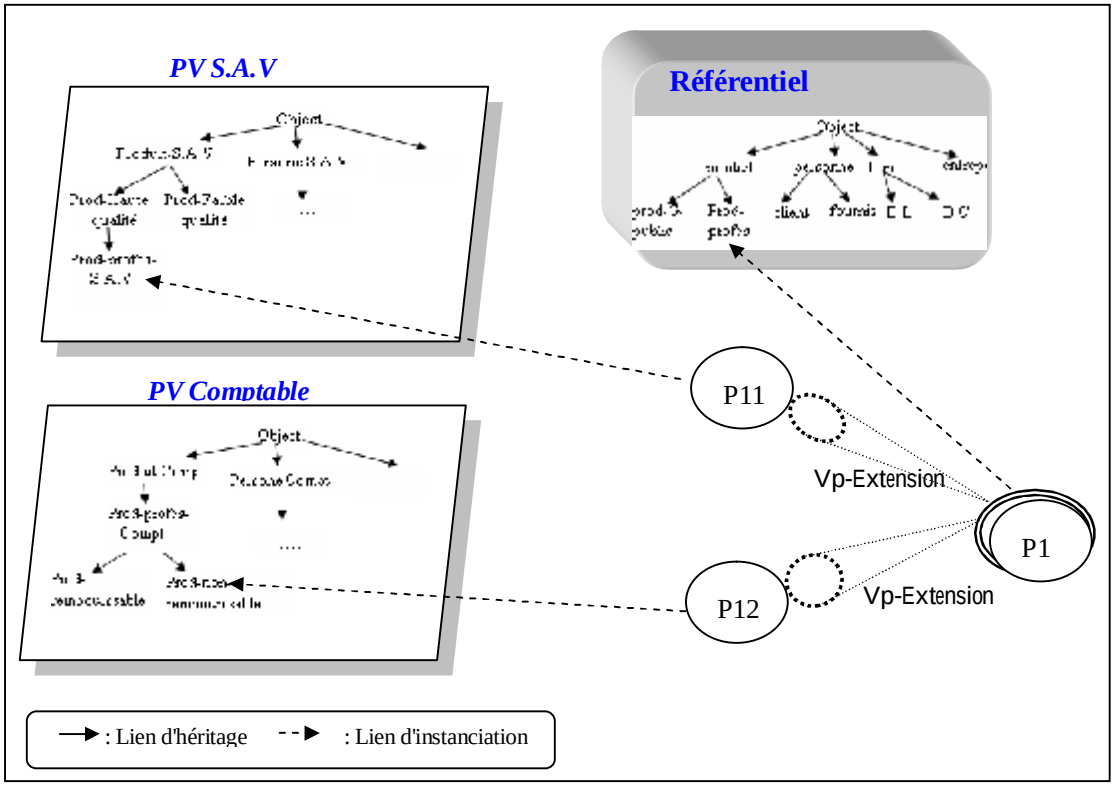


Figure 3. La représentation d'un objet multi-points de vue

1.2.3. Les bases d'objets

Une base d'objets représente l'ensemble des objets d'une base de données dont la structure est exprimée dans le schéma. Dans MVDB on distingue : la base d'objets multi-points de vue et les bases d'objets points de vue.

1.2.3.1. Base multi-point de vue

Une base d'objets multi-points de vue est constituée de l'ensemble des objets multi-points de vue définis ci-dessus. A l'instar de cette définition, une base multi-points de vue est également associée à plusieurs états :

- un état qui regroupe l'ensemble des instances d'objets du référentiel,
- des états définis par les ensembles d'objets points de vue. Chaque ensemble regroupe une représentation partielle des objets du référentiel et constitue une base d'objets points de vue.

1.2.3.2. Base point de vue

Une base point de vue est une représentation partielle associée à une base multi-points de vue dont elle est liée par la relation Vp-Extension. Celle-ci est définie au §2.2.3.

La figure 4 présente une représentation graphique de la base multi-points de vue du schéma multi-points de vue de la figure 2.

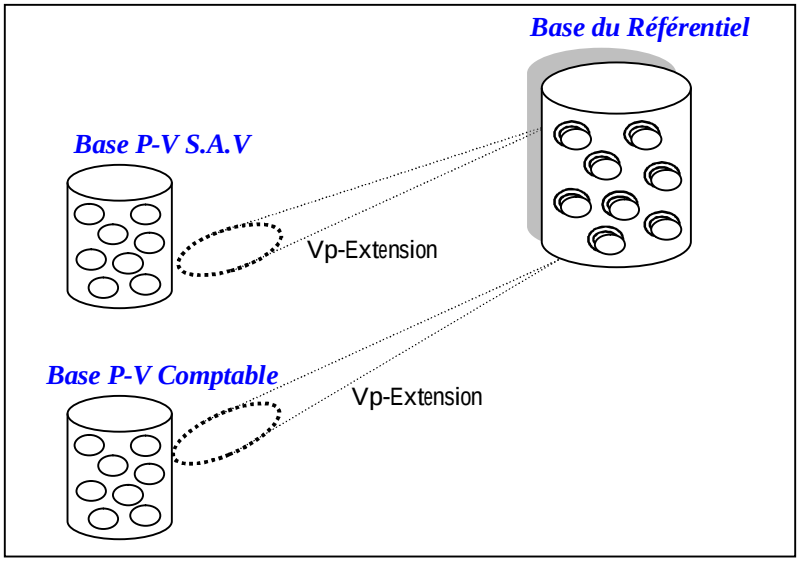


Figure4. La représentation d'une base multi-points de vue

1.3. Tableau récapitulatif des concepts introduits

Nous concluons cette partie par un tableau récapitulatif de la liste des concepts introduits dans notre modèle. Pour chacun de ces concepts, ce tableau décrit le nom, l'élément de modélisation associé et une description informelle de sa sémantique.

Concept	Elément de modélisation	Sémantique informelle
Référentiel (Mvp-Schema)	Hiérarchie de classes	Hiérarchie de classes décrivant la représentation de base des objets
Schéma point de vue (Vp-Schema)	Hiérarchie de classes	Hiérarchie de classes représentant une description partielle des objets selon un point de vue.
Objet multi-point de vue (Mvp-Object)	Objet	Objet ayant une description de base et plusieurs descriptions point de vue.
Objet point de vue (Vp-Object)	Objet	Objet dans un schéma point de vue
Base multi-point de vue (Mvp-Base)	Base	Ensemble des objets du référentiel.
Base point de vue	Base	Ensemble des objets dans un

(Vp-Base)		schéma point de vue.
Vp-Extension	Dépendance	Relation entre les classes étendues du référentiel et celles qui l'étendent dans un schéma point de vue.
Vp-Dependency	Dépendance	Relation entre les points de vue. Des informations du point de vue source dépendent d'autres informations d'un point de vue cible.

Tableau 1. Les concepts du modèle MVDB

2. Description formelle de MVDB

Dans le modèle objet à base de classes, les entités du monde réel ayant une existence propre sont représentées par des objets ou instances d'objets. Chaque objet possède un état, ce sont les variables d'instances ou les attributs de l'objet, et un comportement traduit par un ensemble de méthodes invoquées par envoi de message à l'objet. Nous adoptons un modèle objet à forte encapsulation des données. Ainsi, toute modification d'une variable d'instance est effectuée via une *méthode d'accès*. Une méthode d'accès est une méthode qui permet la mise à jour et la recherche d'une variable d'instance. L'ensemble des variables d'instances et des méthodes d'un objet présentent ses *propriétés*. Tout objet possède un identificateur (OID pour l'anglais *object identifier*). Conceptuellement un OID unique est affecté à tout objet représenté dans la base de données; cette association entre OID et objet reste fixe, même lorsque des attributs de l'objet changent de valeur. L'utilisation des objets et des OID permet aux BDDOO de partager l'information de façon élégante.

Dans une BDDOO, une valeur complexe est associée à tout objet. Cette valeur complexe peut prendre en compte des valeurs affichables et/ou des OID (c'est-à-dire faire référence au même objet ou à d'autres objets).

Les objets qui ont des valeurs complexes de même type peuvent être regroupés en classes. Les classes sont organisées en une hiérarchie fondée sur ce qui est appelé la relation de ou relation de classe à sous-classe. Suivant la tradition des langages de programmation objet, une classe virtuelle **any** est incluse, servant de racine à la hiérarchie. La relation de généralisation/spécialisation obéit à une restriction naturelle fondée sur le sous-typage défini formellement plus loin. Dans ce qui suit nous définissons d'abord le formalisme du modèle objet sur lequel nous nous basons. Ce formalisme est fortement influencé par les modèles IQL et O2 et dont les caractéristiques sont partagées par beaucoup d'autres modèles de BDDOO dont ODMG. Ensuite nous abordons les extensions apportées par notre modèle pour l'intégration du concept de point de vue.

2.1. Concepts et principes du modèle objet de base

La description formelle des concepts de base est inspirée de [1][92] et s'appliquent à toutes les constructions du modèle ODMG. Nous commençons par la définition des aspects structurels (valeur, type ..) et la hiérarchie des classes, puis nous définissons les concepts de schéma et instance de schéma de BDDOO.

Les aspects structurels concernent les notions structurelles de type et d'hiérarchie de classes pour décrire la structure des objets. Des définitions formelles de ces derniers nous permettent de définir un schéma de base de données objets. Pour commencer, nous considérons les ensembles suivants:

- Dom est l'ensemble formé de l'union des domaines des types atomiques de base, disjoints deux à deux : integer, string, bool, et float (pour *entier*, *mot*, *booléen* et *réel*). Les éléments de Dom sont des constantes. Dom = Dom(integer) ∪ Dom(string) ∪ Dom(bool) ∪ Dom(float).
- Class est l'ensemble infini des noms de classes.
- Att est l'ensemble infini des noms d'attributs.
- Obj est l'ensemble infini des identificateurs d'objets.
- **nil** est un symbole qui indique la valeur indéfinie.

2.1.1. Valeurs et objets

Un objet est constitué d'un couple (*identificateur, valeur*). Les identificateurs sont d'un type particulier, contenant les OID. Les valeurs sont des valeurs complexes standard, sauf que les OID peuvent y intervenir.

Définition 1. (*valeur*).
Etant donné O un sous-ensemble de Obj, l'ensemble des valeurs sur O, noté Val(O) est défini comme suit :

(a) **nil** ∈ Val(O);

(b) Dom ⊂ Val(O).

(c) O ⊂ Val(O).

(d) ∀ v1,…, vn ∈ Val(O), ∀ a1, …, an ∈ Att alors <a1 : v1,…, an : vn> ∈ Val(O) et est une valeur structurée de type tuple. Tous les ai sont distincts.

(e) ∀ v1,…, vn ∈ Val(O), alors {v1,…,vn} ∈ Val(O) et est une valeur structurée de type ensemble. {} dénote l'ensemble vide.

(f) ∀ v1,…, vn ∈ Val(O), alors [v1,…,vn] ∈ Val(O) et est une valeur structurée de type liste. [ ] dénote la liste vide.

Définition 2. (*objet*).
Un objet est un couple (o, v), où o est un OID et v sa valeur, appelée également son état.

2.1.2. Types et hiérarchie de classes

Comme vu ci-dessus, les objets sont regroupés en classes. Tous les objets d'une classe ont des valeurs complexes de même type. Le type correspondant à chaque classe

est défini par le schéma de BDDOO. Les types sont définis relativement à un ensemble C donné de noms de classes.

Définition 3. (*type*).
Soit C un sous-ensemble fini de Class. L'ensemble des types qu'on peut construire à partir de C, dénoté Types(C), est défini comme suit :

(a) **any** $\in$ Types(C). (**any** est le type le plus général mais il ne peut pas apparaître à l'intérieur d'un autre type)

(b) C $\subset$ Types(C). Les noms des classes sont des types.

(c) **Integer, String, bool, Float** $\subset$ Types(C).

(d) Si $\tau \in$ Types(C), alors $\{\tau\} \in$ Types(C). $\{\tau\}$ est un type ensemble.

(e) Si $\tau \in$ Types(C), alors $[\tau] \in$ Types(C). $[\tau]$ est un type liste.

(f) Si $\tau 1,\ldots,\tau n \in$ Types(C) et si a1, …, an $\in$ Att, alors <a1 : $\tau 1$,…, an : τn> $\in$ Types(C). <a1 : $\tau 1$,…, an : τn> est un type tuple.

D'après la description informelle, un schéma de BDDOO est une hiérarchie de classes. Celle-ci possède trois composantes : un ensemble de classes, les types associés à ces classes et une définition de la relation entre les classes. Formellement, une hiérarchie de classes est définie par :

Définition 4. (*Hiérarchie de classes*).
Une hiérarchie de classes est un triplet (C, ∂, **p**)
où :

1. C est un sous-ensemble de classes distinctes de Class.
2. ∂ est une fonction de C dans Types(C) qui associe à chaque classe ci de C son type.
3. **p** est une relation d'un ordre partiel sur C. **p** est la relation de spécialisation implantée par un mécanisme d'héritage.

Dans une hiérarchie de classes, le type associé à une sous-classe doit être un raffinement du type associé à sa super-classe. Pour formaliser cette notion, une relation de sous-typage $\leq_t$ est définie.

Définition 5. (*sous-typage*).
Soit (C, ∂, **p**) une hiérarchie de classes. La relation de sous-typage est la plus petite relation d'ordre partiel sur Types(C) qui satisfait les conditions suivantes :

(a) $\forall \tau \in$ Types(C), $\tau \leq_t$ **any** (c'est-à-dire que **any** est au sommet de la hiérarchie).

(b) $\forall$ c, c' $\in$ C, si c **p** c' alors c $\leq_t$ c'.

(c) $\forall \tau 1,\ldots,\tau m, \tau' 1,\ldots,\tau' n \in$ Types(C) et $\forall$ a1, …, an, …am $\in$ Att, si n $\leq$ m et si $\tau i \leq_t \tau' i$ pour tout i $\in$ [1,n] alors <a1 : $\tau 1$,…, an : τn, … am : τm > $\leq_t$ <a1 : $\tau' 1$,…, an : $\tau' n$>.

(d) $\forall \tau, \tau' \in$ Types(C), si $\tau \leq_t \tau'$ alors $\{\tau\} \leq_t \{\tau'\}$.

(e) $\forall \tau, \tau' \in$ Types(C), si $\tau \leq_t \tau'$ alors $[\tau] \leq_t [\tau']$.

Définition 6. (*hiérarchie bien formée*).
Une hiérarchie de classes est *bien formée* si : $\forall$ c, c' $\in$ C, si c **p** c' $\Rightarrow \partial(c) \leq_t \partial(c')$.

2.1.3. Sémantique structurelle d'une hiérarchie de classes

Nous décrivons maintenant la sémantique des classes et des types afin de leur associer des valeurs. Cependant, puisque les valeurs d'une instance de BDDOO peuvent inclure des OID nous commençons d'abord par définir la fonction d'affectation d'OID notée π. Celle-ci affecte un ensemble d'OID à chaque classe.

Définition 7. (*affectation d'OID ou instanciation*).
Soit (C, ∂, **p**) une hiérarchie de classes bien formée. L'affectation d'OID est une application π définie par :
π : C$\rightarrow$ P(O) tel que P est l'ensemble des parties de O qui lie chaque classe à un ensemble fini et disjoint d'OID.
Etant donnée une affectation d'OID π, l'*extension disjointe* de c $\in$ C est π(c) tel que $\forall$ oi $\in$ π(c) $\Rightarrow \nexists$ c' $\neq$ c/ oi $\in \pi$(c'). L'*extension* de c, notée π*(c) rend l'ensemble des objets de C et de toutes ses sous-classes, c'est-à-dire $\pi^*(c) = \cup_{c' \in C} \{ \pi(c')/ c' \mathbf{p} c\}$.

Définissons maintenant la sémantique pour les types relativement à la hiérarchie des classes (C, ∂, **p**) et d'une affectation d'OID π.

Définition 8. (*interprétation de type*).
Soit $O = \cup \{ \pi(c) / c \in C\}$. L'interprétation disjointe d'un type τ, notée Dom(τ) est définie de la manière suivante :

(a) pour chaque type atomique τ, Dom(τ) est l'interprétation usuelle de ce type.

(b) Dom(**any**) = **Val**(O).

(c) $\forall$ c$\in$ C, Dom(c) = $\pi^*(c) \cup \{nil\}$. L'interprétation d'un type de classe est son extension.

(d) Dom({τ}) = {{v1,…, vn} / n $\geq$0 et vi $\in$ Dom(τ), i $\in$ [1..n]}.

(e) Dom([τ]) = {[v1,…, vn] / n $\geq$0 et vi $\in$ Dom(τ), i $\in$ [1..n]}.

(f) Dom(<a1 : τ1,…, ak : τk>) = {<a1 : v1,…, ak : vk>} / vi $\in$ Dom(τi), i $\in$ [1..k].

2.1.4. Schéma de base de données

Nous concluons cette section en présentant les définitions d'un schéma et d'une instance de schéma de bases de données objets. Un schéma est composé d'une hiérarchie de classes bien formée et de la signature des méthodes. L'utilisation des valeurs nommées est très fréquente dans beaucoup de modèles de BDDOO. Une valeur nommée est une structure complexe ayant un type et permet le stockage de valeurs complexes. Ainsi, un schéma peut inclure un ensemble de noms de valeurs avec des types associés. Les méthodes, les langages de programmation et les langages de requêtes peuvent tous utiliser ces noms (pour faire référence à leurs valeurs actuelles) et utiliser les noms de classes (pour faire référence à l'ensemble des objets appartenant à ces classes). Les valeurs nommées et les noms de classes sont analogues aux noms de relations du modèle relationnel.

Définition 11. (*schéma de BDDOO*).
Un schéma de BDDOO est une hiérarchie de classes comme présentée en Définition 4.

2.1.5. Instance de schéma de base de données

Une instance d'un schéma de BDDOO conserve d'une façon extensionnelle les OID des classes, les valeurs des objets et les valeurs des noms persistants et affecte une sémantique aux signatures des méthodes. Cette sémantique est définie par des parties de code. Cependant nous ignorons ici le code des méthodes.

Définition 12. (*instance d'un schéma*).
Une instance d'un schéma encore appelée base est associée à un schéma $S = (C, \partial, \mathbf{p})$. Elle est spécifiée par le triplet $I = (\pi, \nu, O)$, où:

(a) π est la fonction d'extension disjointe.

(b) $O = \cup \{ \pi(c) / c \in C \}$.

(c) $\nu : O \rightarrow \mathbf{Val}(O)$ est une application qui assigne une valeur (un état) à chaque objet de la base. Cette valeur doit être compatible avec le type de la classe de l'objet, i.e. $\forall c \in C, \forall oi \in \pi(c), \mathbf{Val}(oi) \in Dom(\sigma(c))$. Sinon **Val**(oi) n'est pas définie.

2.2. Extensions pour le support du mécanisme de point de vue

Après la définition des principaux concepts du modèle objet sur lequel se base MVDB, nous présentons maintenant les concepts apportés par notre modèle pour supporter les points de vue. Nous commençons par définir la notion de point de vue. Ensuite nous introduisons les notions de référentiel et de schéma point de vue. Ce dernier est une extension de la description de tout ou d'une partie du référentiel selon un point de vue donné. Nous verrons que la construction du schéma point de vue induit quelques conditions sur les types de classes des deux hiérarchies du référentiel et celle du schéma point de vue et sur les liens de généralisation/spécialisation entre les classes du schéma point de vue. Des extensions sont également apportées aux objets. Le concept de référent global et référent local sont introduits afin de définir un objet au niveau local, i.e. au niveau du schéma point de vue, et au niveau global, i.e. au niveau d'un schéma multi-points de vue, respectivement. Les notions d'instances de schémas point de vue et de schéma multi-points de vue sont finalement définies.

MVDB (Sr, VP, C, O) est la spécification de la signature du modèle, c'est une extension du modèle objets à base de classes présenté dans la section précédente. où :

(a) Sr est le nom du référentiel.

(b) VP est l'ensemble infini des noms des points de vue.

(c) C est l'ensemble infini de toutes les classes.

(d) O est l'ensemble infini de tous les objets.

Nous utilisons évidemment les ensembles précédemment définis :

- Dom est l'ensemble formé de l'union des domaines des types atomiques de base, disjoints deux à deux : integer, string, bool, et float (pour *entier, mot, booléen* et

réel). Les éléments de Dom sont des constantes. Dom = Dom(integer) ∪ Dom(string) ∪ Dom(bool) ∪Dom(float).

- Att l'ensemble infini des noms d'attributs.

Et nous spécifions des ensembles spécifiques pour la gestion des points de vue :

- Svp : le nom d'un schéma point de vue tel que : Svp ⊂ VP.
- Bvp : l'ensemble des instances d'un schéma point de vue.
- Cvp : l'ensemble des classes relatives à un schéma point de vue, tel que Cvp ⊂ C.
- Ovp : l'ensemble des objets décrits selon un point de vue, tel que Ovp ⊂ O.

Définition 13. (*point de vue*).
Un point de vue sur un univers de discours est défini par le couple (vp, Svp, Bpv), où :

(a) vp est le nom du point de vue.

(b) Svp est le nom du schéma point de vue.

(c) Bvp est le nom de la base du schéma point de vue.

Nous notons que :

- $\cup_{vp \in VP} \{Ovp\} \subset O$
- $\cup_{vp \in VP} \{Cvp\} \subset C$

2.2.1. Les schémas dans MVDB

En se basant sur les définitions formelles de la section précédente, nous présentons la définition des schémas dans le modèle MVDB qui sont de deux types : le schéma référentiel et les schémas points de vue.

2.2.1.1. Schéma référentiel

Nous considérons que le schéma d'une base de données multi-points de vue est défini via la définition du référentiel.

Définition 14. (*schéma référentiel*).
Un schéma référentiel est le schéma de base sur lequel s'appuie la définition de toute représentation partielle des objets selon un point de vue donné. C'est un schéma dont la hiérarchie, à l'instar de la définition 4, est (Cr, ∂, **p**) où Cr est l'ensemble des classes du référentiel, tel que Cr ⊂ C.

2.2.1.2. Expression du schéma référentiel en ODL

ODL permet l'expression d'un schéma de bases de données objet en utilisant la notion de classe ou d'interface (voir description détaillée en annexe B). Dans le contexte de notre travail, toutes les entités sont décrites par des classes ODL. Une partie du schéma référentiel de notre application exprimé en langage ODL est présenté en figure 5. La description complète du référentiel se trouve en annexe A. Cependant, nous remarquons :

- La notion de clé est analogue à celle du modèle relationnel.
- La clause **relationship** permet de définir l'extrémité d'une relation. Elle consiste en un attribut dont le domaine est le type de l'autre extrémité. Notons ici, qu'une

relation n'est pas une construction nouvelle apportée au modèle "objet". Il s'agit d'un attribut dont le type est un ensemble de littéraux.

```
class  Personne (extent personnes key nom, telephone) {
      attribute string nom;
      attribute string telephone;
      attribute string adresse;
}

class  client : Personne(extent clients key nom, telephone) {
      relationship set<Produit> a_achete inverse Produit::a_été_acheté_par;
}

class  fournisseur : Personne(extent fournisseurs key nom, telephone) {
      relationship set<Produit> fournit inverse Produit::est_fourni-par;
}

class  Produit (extent produits) {
      attribute string designation;
      attribute integer prix_unitaire;
      relationship set<Client> a_été_acheté_par inverse Client:: a_achete est;
      relationship set<Fournisseur> est_fourni-par inverse Fournisseur:: fournit;
}

class  Produit_professionnel : Produit (extent produits_prof
                                        key reference_professionnelle) {
      attribute string reference_professionnelle;
}

class  Produit_grand_public :Produit (extent produits_grand_public
                                      key reference_grand_public) {
      attribute string reference_grand_public;
}
…….
```

Figure 5. Le schéma référentiel de l'entreprise de fabrication et de distribution des produits spécifié en ODL

2.2.1.3. Schéma point de vue

La définition d'un schéma point de vue se base sur deux opérations : **la projection** et **l'extension**.

Définition 15. (*la projection*).
Soit S=(C, ∂, **p**) et S'=(C', ∂', **p**') deux schémas. S' est une projection de S, noté S'∇S ou S'=∇(S), si les conditions suivantes sont vérifiées:

(a) C'$\subseteq$C.

(b) $\partial'|_C = \partial$ ($\partial'|_C$ est la restriction de ∂' dans S' à l'ensemble des classes C de S).

(c) **p**'$|_C$ = **p** (**p**'$|_C$ est la restriction de **p**' dans S' à l'ensemble des classes C de S).

La projection d'une hiérarchie S est une hiérarchie S' composée d'un sous-ensemble des classes de S. Celles-ci préservent leur type (propriété (b)) et leur lien de spécialisation/généralisation (propriété (c)) dans S.

Propriété du schéma projeté

Une propriété importante que doit vérifier le schéma projeté S' sur S est la *fermeture*. En effet, S' est construit à partir d'une sélection explicite des classes de S. La fermeture concerne les attributs des classes de S dont le domaine est un type autre qu'un littéral (ie. une classe) et consiste à s'assurer que toute classe qui représente le domaine d'un des attributs des classes sélectionnées dans S soit présente dans S'. Pour donner une définition formelle de la fermeture de S', nous considérons la fonction "Uses". "Uses" est une fonction de C' $\rightarrow$ $2^{C'}$ définie par : Soit $c \in C'$, $Uses(c) = \{c_1 \in C / \exists a \in att(c)$ et $Type_a(c) = c_1\}$ et permet donc d'identifier l'ensemble de classes qui sont des types d'attributs d'une classe. "Uses" est définie sur le schéma S' par : $Uses(S') = \bigcup_{c \in C'} Uses(c)$. $Uses^*$ est ensuite définie par :
$Uses^*(S') = \bigcup_{j=1,|S'|} Uses(S')$ où pour $j>1$ $Uses^j(S') = Uses^{j-1}(S')$ et $Uses^1(S') = Uses(S')$.

Définition 16 (schéma fermé).
Le schéma S' est fermé si et seulement si $S' = C' \cup Uses(S')$

Dans le modèle MVDB, on vérifie que le schéma projeté construit par l'utilisateur est fermé. Si ce n'est pas le cas, on calcule $Uses^*(S')$ afin de lui adjoindre les classes nécessaires pour assurer sa fermeture.

La figure 6 montre un exemple de schéma non fermé ainsi que l'ensemble minimal des classes nécessaires pour le fermer. Chaque flèche représente un attribut dont le domaine est le type de la classe pointée.

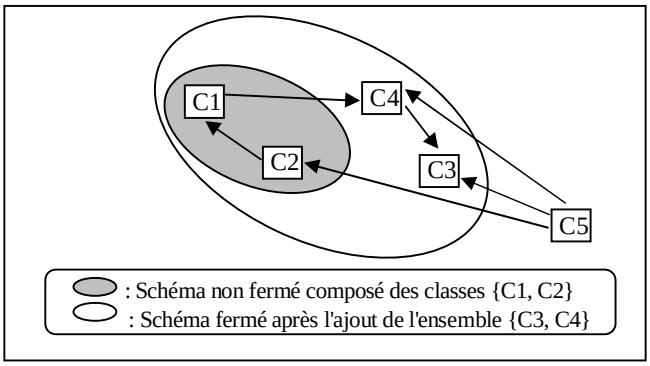


Figure 6. Exemple de schéma non fermé et de l'ensemble minimal des classes à ajouter pour assurer sa fermeture

Définition 17 (*l'extension*).
Soit $S' = (C', \partial', \mathbf{p}')$ et $S = (C, \partial, \mathbf{p})$ deux schémas. L'extension de S par S', noté $S' \Delta S$ ou $S' = \Delta(S)$, est définie par :

1. $C' = CE \cup CA$ avec :
   - $CE \subseteq C$. CE est l'ensemble des classes de S étendues dans S'.
   - CA est l'ensemble des classes ajoutées dans S'.
2. Une relation d'extension Vp-Extension est définie sur CE pour assurer la bonne construction de la hiérarchie S' relativement à S. Les propriétés de cette relation sont :

a. $\forall c_1 \in C, \exists c'_1 \in CE/$ Vp-Extension $(c_1, c'_1) \Rightarrow \partial(c_1)$ **p** $=' \partial'(c_1')$.

b. $\forall c'_1 \in CE, c_1, c_2 \in C$, Vp-Extension $(c_1, c'_1) \wedge$ Vp-Extension $(c_2, c'_1) \Rightarrow c_1 = c_2$.

c. $\forall c'_1, c'_2 \in CE, c_1, c_2 \in C$, Si c'_1 **p** $' c'_2$, Vp-Extension $(c_1, c'_1) \wedge$ Vp-Extension $(c_2, c'_2) \Rightarrow c_1$ **p** c_2.

L'extension d'une hiérarchie S par une hiérarchie S' s'effectue à deux niveaux:

- Les types des classes de S peuvent être étendus dans S' par ajout de nouveaux attributs,
- S est étendue dans S' par la création de nouvelles classes.

Les deux types d'extension ne doivent pas remettre en cause les liens hiérarchiques des classes dans S. Cependant des règles de bonne construction de la hiérarchie S' sont à respecter (voir explication plus bas)

Des deux définitions précédentes nous obtenons celle d'un schéma point de vue.

Définition 18. (*schéma point de vue*).
Un schéma point de vue Svp=(Cvp, ∂ vp, **p** vp) est une *extension* de la *projection* sur le schéma du référentiel Sr=(Cr, ∂ r, **p** r). Svp= $\Delta(\nabla(Sr))$ (voir Figure 7).

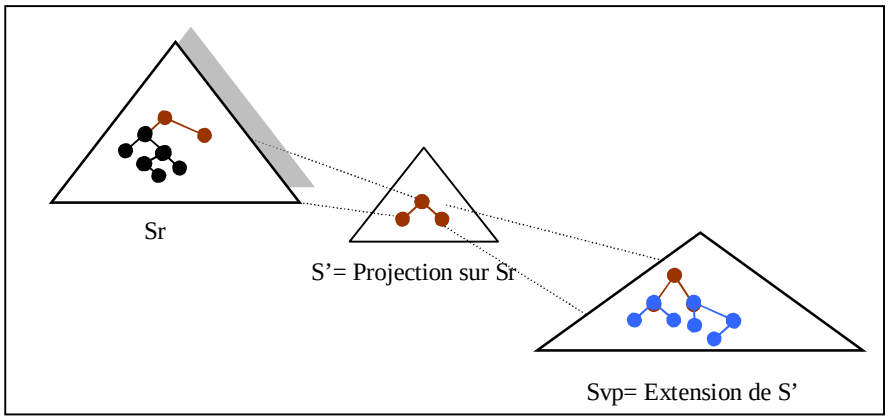


Figure 7. Schéma point de vue = Projection + Extension du référentiel

La construction d'un schéma point de vue consiste à étendre une hiérarchie qui résulte de la projection sur le schéma référentiel. Cette hiérarchie, notée S' dans la figure 7 est composée d'un sous-ensemble (voir tout l'ensemble) des classes du référentiel appelées les classes étendues (CE). Cette construction est restrictive comparée à celle d'une hiérarchie de classes classique. En effet, les classes CE sont (si nécessaire) d'abord adaptées aux spécificités du point de vue. Cette adaptation est régie par la relation Vp-Extension (voir propriété 2).

En effet, Par cette relation, toute classe de C ne peut être étendue que par une seule classe de CE (Propriété 2.b). Cette extension peut éventuellement enrichir la description de base de cette classe (Propriété 2.a). Les liens de généralisation/spécialisation entre les classes importés sont préservés dans CE et par conséquent la relation d'héritage. Cependant, l'extension d'une classe importée dans un

schéma point de vue est héritée par ses sous-classes. Une conséquence de cet héritage est qu'une sous-classe d'une classe enrichie est également enrichie dans ce point de vue. Elle l'est soit explicitement par l'extension de sa structure comme la classe "Produit" de la figure 8, soit implicitement par l'héritage de l'extension de sa superclasse comme la classe "Prod_grand_public". L'enrichissement implicite provoque la participation implicite de l'entité correspondante dans le point de vue correspondant. Enfin, la hiérarchie des classes importées peut éventuellement être étendue par la création de nouvelles classes dites classes ajoutées (CA). Celles-ci permettent le raffinement de S' selon un point de vue donné. L'ensemble des classes d'un point de vue Cvp est : Cvp= CE ∪ CA.

2.2.1.4. Expression du schéma référentiel en ODL

Dans le modèle ODMG, pour assurer la fermeture du schéma projeté à partir des classes du référentiel, on doit vérifier que toute relation définie dans une classe importée est totalement partagée. En d'autres termes, il faut que les classes situées aux deux extrémités d'une relation fassent partie de l'ensemble des classes projetées. Par exemple si la classe Produit est décrite selon le point de vue "Comptable" alors les classes Client et Fournisseur le sont également.

Cependant, pour représenter une telle extension, nous introduisons une nouvelle forme syntaxique d'"extension" de classe (**extend**) en enrichissant ODL pour nos besoins. Cependant, l'extension "**extend"** ne doit pas être confondue avec "**extent**" qui désigne la variable des instances de la classe. L'extension proposée est :

```
class <viewpoint_class_name> extend <referential_class_name>
 {
   viewpoint-part:
       [<attribute_definition>]*
       [<relationship_definition>]*
   referentiel-part:
       [<attribute_definition>]*
       [<relationship_definition>]*
}
```

Avec:

- <viewpoint_ class_name> est le nom de la classe à étendre selon un point de vue.

La figure 8 présente l'extension du référentiel de la figure 6 selon le point de vue "Comptable". Les classes du référentiel concernées par la projection sont toutes les entités du domaine de notre application sauf la classe "Prod_grand_public" et la classe "Personne". Les classes de la projection sont étendues par l'adjonction de nouvelles propriétés, exemple "encours_a_régler" et "chiffres_affaires" dans la classe "Client" et "sommes_aquitées" et "sommes_dues" dans la classe "fournisseur". La classe des produits professionnels "Produit_professionnel" est spécialisée par deux nouvelles classes: la classe "Produit_rembourssable" et la classe "Produit_non_rembourssable".

```
class  Part_externe_comptable extend Personne (extent personnes_comptable key nom, telephone) {
      referentiel part :
            attribute string nom;
            attribute string telephone;
            attribute string adresse;
}
class client_ comptable: Part_externe_ comptable extend Client (extent clients_ comptable key nom,
telephone)
{
    viewpoint part :
      attribute integer  en_cours_a_regler;
      attribute integer chiffre_affaire;
   referentiel part :
      relationship set<Produit_comptable>a_achete inverse Produit_ comptable::a_été_acheté_par;
}
class  fournisseur_comptable Part_externe_comptable extend fournisseur (extent
                                                        fournisseurs_comptable  key nom, telephone)
{
    viewpoint part :
      attribute integer  sommes_aquitées;
      attribute integer sommes_dues;
   referentiel part :
      relationship set<Produit_comptable> fournit inverse Produit_ comptable::est_fourni-par;
}
class  Produit_ comptable extend Produit (extent produits)
{
    viewpoint part :
      attribute integer  marge_moyenne;
   referentiel part :
      attribute string designation;
      attribute integer prix_unitaire;
      relationship set<Client_comptable> a_été_acheté_par inverse Client_ comptable:: a_achete est;
      relationship set<Fournisseur_comptable> est_fourni-par inverse Fournisseur_ comptable:: fournit;
}
class  Produit_professionnel_comptable:Produit_comptable extend Produit_professionnel (extent
                                        produits_prof_comptable key reference_professionnelle) {
    referentiel part :
      attribute string reference_professionnelle;
}
class  Produit_rebourssable :  Produit_professionnel_comptable (extent produits_rembourssables
                                key reference_professionnelle) {
      attribute string garantie;
}
class  Produit_non_rebourssable :  Produit_professionnel_comptable
      (extent produits_non_rembourssables  key reference_professionnelle) {
      attribute string delai;
}…….
```

Figure 8. La représentation de l'entreprise de fabrication et de distribution des produits selon le point de vue "Comptable"

Il importe de remarquer ici que toute "relationship" (relation entre deux entités) introduite dans le référentiel est partagée totalement dans un schéma point de vue[2] : c'est le principe de la fermeture d'un schéma présentée en Définition 16. De ce fait, si une telle relation est partagée par un point de vue, il faut que les entités situées aux deux extrémités de la relation participent à ce point de vue. Par conséquent, les entités situées aux deux extrémités d'une relation introduite dans le référentiel doivent participer aux mêmes points de vue et leurs classes doivent donc être présentes dans les schémas points de vue correspondants.

Par exemple, les relations entre les entités "Produit" et "Fournisseur" et entre les entités "Produit" et "Client" sont introduites dans le référentiel. Dans la figure 8, on vérifie que ces entités sont toutes présentes dans le schéma du point de vue "Comptable". Il est facile de vérifier cette propriété dans les différents schémas point de vue de notre application de fabrication et de gestion des produits.

2.2.2. Les objets

Les différents schémas (schéma référentiel et les schémas points de vue) détiennent tous les objets persistants de la base de données orientée objet. Ces objets sont les instances des différentes classes. Avant de décrire formellement les bases de ces schémas, nous présentons d'abord les extensions apportées au concept d'objet.

Dans le modèle objet de référence, un objet correspond au couple (o, v), où o est l'identifiant de l'objet et v sa valeur. Dans cette présentation, tout objet est instance directe d'une seule classe, c'est le principe de l'instanciation. Cette assertion est restrictive et pose des problèmes qu'en à la modélisation de la dynamique et de la représentation multiple des objets du monde réel. La modélisation de la dynamique et de la représentation multiple des objets sont des caractéristiques requises dans l'étude avancée de la technologie orientée objet. En effet, le mécanisme de l'instanciation stricte (unique et figée) a été assoupli par l'instanciation multiple par laquelle on permet à un objet d'être instance directe de plus d'une classe d'un schéma orienté objet. Dans le contexte de notre travail, les objets possèdent les propriétés suivantes:

Propriété 1. Un objet est instance dans le schéma référentiel et instance dans un ou plusieurs schémas point de vue.

Un objet possède donc une description de base dans le référentiel et des descriptions partielles selon des points de vue. Cette propriété est inhérente au mécanisme de point de vue. Un objet dans un point de vue est considéré comme une instance de celui-ci, ce qui engendre une description multiple des objets dans un schéma multi-points de vue.

Propriété 2. L'état d'un objet est orienté point de vue.

Cette propriété stipule que l'état d'un objet dépend du point de vue dans lequel il est considéré. Donc, toute description d'un objet selon un point de vue est vue comme une description complète (selon un point de vue) et séparée de l'objet. Ce qui complète d'une manière naturelle la première propriété.

D'après les propriétés précédentes, nous proposons l'introduction d'un nouveau concept appelé *Référent d'objet*. Nous distinguons le *Référent local* et le *Référent global* d'un objet.

[2] Pour respecter le fait que l'on partage la relation avec sa sémantique complète.

Définition 19. (*Référent local*).
Un référent local d'un objet, noté r_l, est l'identification de l'objet dans un schéma point de vue, tel que $r_{l=}(vp, o_l)$ où :

- vp est le nom du schéma point de vue.
- o_l est l'identifiant (OID) de l'objet dans un point de vue.

Définition 20. (*Objet point de vue*).
Un objet point de vue est un couple (r_l , v_l) où v_l est son état au niveau local, i.e. au niveau point de vue.

Un référent local permet à un objet d'être localement géré au niveau d'un schéma point de vue. De ce fait, chaque base de données point de vue préserve son autonomie d'exécution. Cependant, une métabase de données est associée avec cette dernière afin d'assurer la cohérence des données.

Définition 21. (*Référent global*).
Un référent global d'un objet, noté r_g, est l'identification de l'objet dans un schéma multi-points de vue, tel que $r_{g=}(o_g, L(r_l))$ où :

- o_g est l'identifiant (OID) de l'objet dans le référentiel.
- L est la liste des référents locaux de l'objet : $L(r_l) = \cup_{vp \in VP} \{ r_l \}$.

Définition 22. (*Objet multi-point de vue*).
Un objet multi-point de vue est un couple (r_g , v_g) où v_g est son état au niveau du référentiel.

Le référent d'objet est un concept important inhérent à notre modèle. Il permet l'accès aux données dans les schémas point de vue via la liste des référents locaux.

2.2.3. Les bases d'objets

Une base multi-points de vue est constituée de l'ensemble de tous les objets multi-points de vue MVO (MVO $\subset$ O) dans le référentiel à partir duquel on peut définir les autres points de vue. Elle est définie par :

Definition 23. (*Base multi-point de vue*)
Soit R_r l'ensemble infini des référents globaux des objets du schéma référentiel Sr.
Une base multi-point de vue, notée Br, est un état du schéma spécifié par le tuple (π_r, π'_r, O_r, $\sqrt{}_r$) où:

- $\pi_r : C_r \rightarrow P(R_r)$ est la fonction qui associe à chaque classe $c \in C_r$ des référents globaux. C_r est l'ensemble des classes dans Sr.
- O_r est l'ensemble des objets dans Br.
- $\pi'_r: (O_r , R_l) \rightarrow O_r$ est la fonction qui ajoute à un objet de O_r un référent local relatif à un point de vue. $O_r = \uplus_{c \in Cr} \{(\pi_r (c), \uplus_{vp \in VP} (R_l))\}$,
- $\sqrt{}_r$ est la fonction qui associe une valeur à chaque objet de B_r, tel que:
  $\forall c \in C_r , \forall o \in \pi_r(c) , \sqrt{}_r(o) \in Dom(\sigma_r(c))$.

Une base point de vue est une représentation partielle associée à tout ou à un sous-ensemble des objets de la base du référentiel selon un point de vue donné. Donc il s'agit du même ensemble des objets du référentiel qui est représenté dans les différents points de vue. De ce fait, tout objet de la base point de vue est une extension de la description d'un objet du référentiel selon un point de vue donné.

En suivant le même procédé utilisé pour l'obtention d'un schéma point de vue, nous obtenons une base point de vue par extension des objets du référentiel. La relation Vp-Extension est alors définie pour lier un objet point de vue à son objet de base dans le référentiel.

Definition 24. (*Base point de vue*)
Soit B_r (π_r, π'_r, O_r, $\sqrt{}_r$) la base du référentiel de schéma S_r=(C_r, ∂_r, ρ_r) et R_l l'ensemble infini des référents locaux des objets d'un schéma point de vue S_{pv}=(C_{pv}, ∂_{pv}, ρ_{pv}).
B_{vp}(π_{pv}, O_{pv}, $\sqrt{}_{pv}$) est la base point de vue extension de Br selon un point de vue donné si les conditions suivantes sont vérifiées:

1. $O_{vp} \subset O_r$. O_{vp} est l'ensemble des objets dans un point de vue.
2. π'_{pv}: $C_{pv} \rightarrow P(R_g)$ est la fonction qui associe à toute classe point de vue l'ensemble des objets du référentiel qu'elle peut étendre. π'_{pv} est définie sur C_{pv} (C_{pv} = CE $\cup$ CA) par :
   a. Pour $c_2 \in$ CE, $c_1 \in$ Cr avec Vp-Extension (c_1, c_2) on a : $\pi'_{pv}(c_2)$ = {o $\in$ E/ E= $\pi_r^*(c_1)$ - $\cup c_1'(\{\pi_r^*(c_1') / (c_1 \# c_1') \wedge (c_1' <_r c_1) \wedge \exists c_2'$, Vp-Extension ($c_1'$, c_2')}
   b. Pour $c_2 \in$ CA, $\pi'_{pv}(c_2) = \pi'_{pv}(c_2')$/ $c_2' \in$ CE $\wedge$ ($c_2 < c_2'$) $\nexists c_2'' \in$ CE, (c_2' # c_2'') $\wedge$ ($c_2 < c_2'' < c_2'$) $\wedge \exists c_1' \in$ C, Vp-Extension (c_1', c_2'')
3. π_{pv} : $C_{pv} \rightarrow R_l$ est la fonction qui associe à chaque classe $c \in C_{pv}$ des référents locaux. Cependant une relation Vp-Extension permet de lier tout objet point de vue instance directe de C_{pv} à un seul objet du référentiel. Cette relation possède les propriétés suivantes :
   a. $\forall c_2 \in O_{pv}$, $c_1, c_1' \in O_r$, Vp-Extension (o_1, o_2) $\wedge$ Vp-Extension (o_1', o_2) $\Rightarrow o_1 = o_1'$.
   b. $\forall o_2 \in O_{pv}$, $c \in C_{pv}$, Vp-Extension (o_1, o_2) $\wedge$ $o_2 \in \pi_{pv}(c) \Rightarrow o1 \in \pi'_{pv}(c)$.
4. $\sqrt{}_{pv}$ est la fonction qui associe une valeur à chaque objet de B_{pv}, tel que:
$\forall c \in C_{pv}, \forall o \in \pi_{pv}(c)$, $\sqrt{}_{pv}(o) \in Dom(\sigma_{pv}(c))$.

D'après la définition précédente, nous dirons qu'une base point de vue est constituée d'objets qui sont instances des classes du schéma point de vue. Cependant, à chaque instanciation d'objet, l'utilisateur doit préciser à quel objet de base cette instance se rapporte par le biais de la relation Vp-Extension. En effet, cette relation exprime le fait qu'un objet dans une base point de vue est une représentation partielle d'un objet de base du référentiel. Ce dernier est unique et ne peut être associé à un autre objet point de vue de la même base point de vue (Propriété 3.a). Cependant, la relation Vp-Extension induit une condition sur les classes d'instanciation de deux objets o_1 et o_2 appartenant à une base multi-point de vue et une base point de vue respectivement. Ainsi, lorsque l'utilisateur crée un objet point de vue instance directe d'une classe point de vue, le choix de l'objet multi-point de vue n'est pas arbitraire, mais devrait vérifier la propriété 3.b. Par exemple, dans la figure 8, soit l'objet o_2 instance de la classe point de vue "Produit_Comptable". D'après cette propriété et la définition de la fonction π'_{pv} (Définition 23.2), l'objet multi-point de vue qui correspond à o_2 ne peut être que instance directe de la classe multi-point de vue "Produit_grand_public". De même, si o_2

est une instance directe de la classe point de vue "Prod_rembourssable", l'objet multi-point de vue qui lui correspond est peut être que instance directe de la classe multi-point de vue "Prod_professionnel".

2.3. L'évolution de la représentation des objets

La représentation multiple dans MVDB est directement liée à sa représentation évolutive. Tout en respectant le principe de la représentation de base et les contraintes d'intégrité entre points de vue, un objet peut acquérir de nouvelles représentations selon les différents points de vue comme il peut en perdre d'autres. Cette évolution comparable au mécanisme des rôles, largement étudié en bases de données, se traduit par des ajouts et des retraits de liens de représentation entre un objet et les classes d'appartenance dans les schémas points de vue. Cet aspect d'évolution est un atout particulièrement puissant pour l'évolution d'une base de données.

3. Conclusion

Si l'approche objet permet de découper un système d'information en entités distinctes, elle ne permet pas de séparer les différentes descriptions à l'intérieur des entités. L'approche MVDB que nous proposons, ajoute ce découpage et permet à plusieurs experts de décrire un même univers de discours avec pour chacun une vision qui lui est propre, adaptée aux spécificités d'une description particulière des entités. Dans cette approche, le schéma du référentiel, encore appelé le schéma de base, correspond aux éléments communs aux différentes descriptions partielles. Les schémas points de vue quant à eux ajoutent au schéma du référentiel un certain nombre d'éléments spécifiques tout en garantissant la compatibilité et l'intégrité du système global. Une définition formelle des différents concepts introduits pour le support des points de vue a été présentée dans ce chapitre. Dans le chapitre suivant, nous présentons les mécanismes qui permettent de partager des données entre les points de vue et d'assurer la cohérence du système ainsi construit.

Chapitre 4

Où l'on présentera des mécanismes inhérents à notre modèle et qui consistent au partage, l'échange, la protection et la cohérence des données.

En effet, l'échange d'informations entre les points de vue définis à partir d'un même référentiel est un aspect important dans l'approche multi-points de vue. Il importe alors de définir des liens entre les schémas points de vue. Ces liens sont basés sur des relations de dépendances et réalisés par le mécanisme de "passerelle" que nous proposons et définissons formellement. Le partage des données concerne également les objets, un aspect complètement absent du modèle de classes sur lequel se base MVDB. Afin de permettre ce partage, nous empruntons le mécanisme de délégation des langages de prototypes et que nous adoptons à nos besoins.

Enfin, nous étudions une fonctionnalité indispensable et complémentaire à tout modèle: la cohérence. Nous présentons des aspects spécifiques à notre modèle et qui nous ont amené à considérer une typologie de contraintes qui lui sont propres. Un langage d'expression des contraintes ainsi que des fonctions permettant leur évaluation sont définis.

1. Le partage des données dans un schéma multi-points de vue

Dans cette section nous abordons le problème de partage des données dans un schéma de base de données multi-points de vue. En effet, les données décrites dans un schéma point de vue sont soit spécifiques au point de vue correspondant où partagées, i.e. en provenance d'autres ressources. Les données partagées peuvent provenir de deux types de ressources (cf. §1.2.1.3 du chapitre 3) :

- à partir du référentiel de base dont la description est commune à l'ensemble des points de vue,
- à partir d'un ou de plusieurs points de vue qui sont issus du même référentiel. Ce partage donne lieu à un échange ou une transmission des informations pertinentes d'un point de vue fournisseur d'informations à un ou plusieurs points de vue consommateurs.

Le premier type de partage a été abordé et défini dans le chapitre précédent. C'est un aspect inhérent à la construction même des schémas points de vue. Dans ce qui suit nous présentons l'interaction et l'échange de données entre points de vue en proposant une définition formelle de ce mécanisme.

Cependant, les deux types de partage de données posent le problème de la protection de celles-ci puisqu'elles sont définies dans un schéma et exploitées dans d'autres schémas. Quelles conditions doit-on imposer quand à leur manipulation et quelle marge de liberté doit-on accorder pour leur mise à jour là où elles ne sont pas définies ?. Nous présentons cet aspect à travers des exemples tirés de notre application de l'entreprise de gestion des produits et nous présentons le mécanisme qui permet le partage des données tout en assurant leur protection.

1.1. L'interaction entre les schémas points de vue

L'échange de données entre points de vue permet un accès inter-points de vue d'attributs. Cet échange se fait plus exactement entre deux classes appartenant à deux schémas points de vue différents et dont la description est relative à la même entité conceptuelle du domaine. Pour modéliser cet accès, nous avons défini le concept de *passerelle.*

Définition 1. *(Passerelle)*

Une passerelle est un lien ayant un sens précis spécifié par :

- la classe du schéma point de vue fournisseur de données, dite la classe source et notée C_{sr},
- la classe du schéma point de vue consommateur de données, dite la classe destination et notée C_{dst},
- l'ensemble des données échangées noté P. P est un tuple formé d'attributs sélectionnés du type de la classe C_{sr} et qui enrichissent le type de la classe C_{dst}.

Une passerelle permet de mettre en relation de dépendance deux classes de deux schémas points de vue différents. La figure 1 présente la définition de trois passerelles entre les points de vue *Comptable*, *Commercial* et *Direction* pour l'entité *Client.* La passerelle P1 sélectionne l'attribut *encours_a_regler* de la structure de la classe

Client.Comptable pour le partager dans la classe Client-Commercial. La passerelle P2 lie les classes Client-Commercial et Client-Direction via les attributs *date_dernière_cmd*. Enfin, *montant_cmd* et la passerelle P3 dont l'ensemble des attributs est composé de *encours_a_regler* et *chiffre_affaires* permet de lier la classe Client.Comptable à la classe Client-Direction.

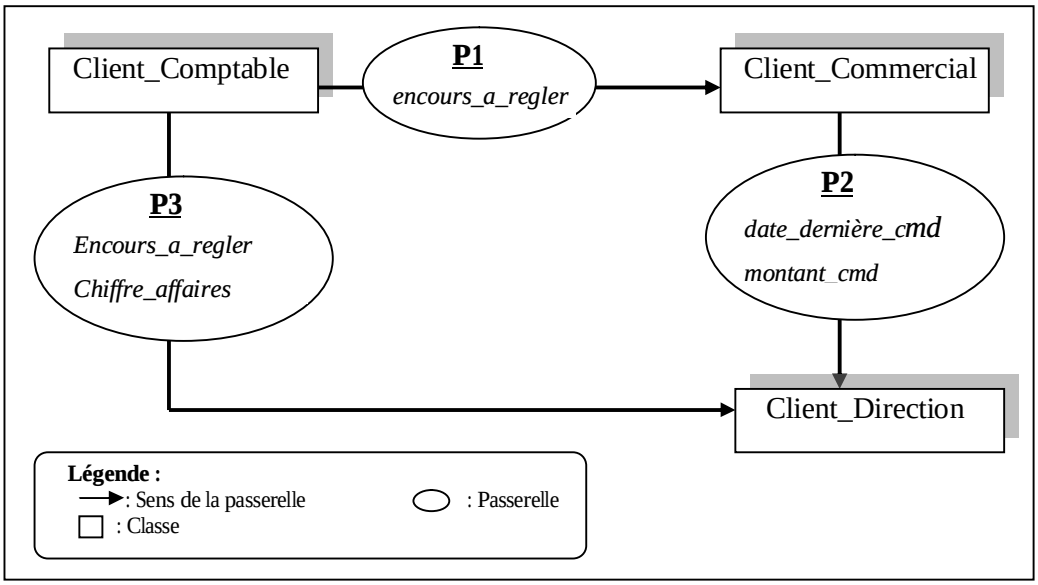


Figure 1. Le partage de données entre les classes de différents points de vue. "Les passerelles"

Nous notons VP-Dependency la relation entre deux classes de deux points de vue différents engendrée par une passerelle. Cette relation est régit par certaines conditions qui assurent une extension cohérente d'une classe par les attributs d'une passerelle.

1.1.1. Définitions formelles

Nous définissons formellement la relation VP-Dependency puis l'extension d'un schéma point de vue par un ensemble de passerelles issues de différents points de vue.

Soit P l'ensemble de toutes les passerelles dans un schéma multi-points de vue de référentiel $Sr = (Cr, \partial r, \mathbf{p} r)$. Soient $Svp_1 = (Cvp_1, \partial vp_1, \mathbf{p} vp_1)$ et $Svp_2 = (Cvp_2, \partial vp_2, \mathbf{p} vp_2)$ deux schémas point de vue définis à partir de Sr. "Att" est la fonction qui renvoie les attributs d'une passerelle.

Définition 2. *(la relation VP-Dependency)*
Soient la classe $C_2 \in Cvp_2$ et la passerelle $P \in P$ définie de $C_1 \in Cvp_1$ vers C_2. C_2 est en dépendance avec C_1 via la passerelle P et on note *VP-Dependency*(C_1, C_2, P), si et seulement si:

1. $\exists C \in Cr$ / *VP-Extension*[1](C, C_1) $\wedge$ *VP-Extension*(C, C_2).
2. $\forall a \in Att(P)$, on a :
   a) $a \notin Types(Cr)$.

[1] VP-Extension est défini au §2.2.1.2. du chapitre 3.

b) $a \in Types(Cvp_2) \wedge \neg (\exists SC \in Cvp_1 \wedge C_1 \mathbf{p} vp_1 SC \wedge a \in \partial vp_1(SC))$.

La relation *VP-Dependency* permet un échange de données entre deux classes appartenant à deux schémas points de vue différents. Ces classes sont liées à une même entité conceptuelle du domaine par la relation *VP-Extension* (condition 1). En effet, nous rappelons que MVDB permet de décrire les entités d'un univers de discours selon différents points de vue, donc l'échange préconisé par notre modèle concerne le partage des données entre les différentes parties descriptives d'une même entité. En plus chaque attribut échangé, qu'il soit de type littéral ou non, doit obligatoirement avoir pour classe d'introduction la classe source de la passerelle. Il ne peut avoir lieu dans l'une des sous-classes (propriété 2.b). Un exemple justifiant cette propriété est présentée au §1.1.3.

Une passerelle permet d'enrichir la classe destination par un ensemble d'attributs. Ce qui engendre automatiquement l'extension de l'état de ses objets par les valeurs de ces attributs. L'extension d'une classe par une passerelle est définie formellement par :

Définition 3. (*Extension d'une classe par une passerelle*)
La dépendance *VP-Dependency*(c_1, c_2, P), permet de produire une classe c'_2 qui est une extension de c_2 par la passerelle P. L'intension de c'_2 est : att(c'2) = att(c1) + att(P)

1.1.2. Propriétés

Les passerelles possèdent deux propriétés importantes.

Propriété 1.
$\forall P_1, P_2 \in P, C_{dst}(P_1) = C_{sr}(P_2) \Rightarrow Att(P_1) \mathbf{I} Att(P_2) = \varnothing$

Cette propriété vient pour compléter la condition 2 de la définition d'une passerelle. En fait, pour une classe source C_{sr}, tout attribut défini dans une passerelle P_1 ne doit pas provenir d'une définition préalable d'une autre passerelle P_2 dont C_{sr} est une classe destination. En outre, cette propriété évite à une classe destinataire d'une passerelle de sélectionner dans une passerelle un attribut d'une autre classe, initialement lui était propre, ce qui n'a pas de sens.

Propriété 2.
Dans un schéma point de vue $Svp_2 = (Cvp_2, \partial vp_2, \mathbf{p} vp_2)$, et pour toute classe $C_2 \in Cvp_2$, les attributs de C_2 qui sont issus de passerelles sont héritées par ses sous-classes. En fait :
$\forall C_1, C_2 \in (Cvp_1, Cvp_2) \wedge P \in P \wedge$ *VP-Dependency*(C_1, C_2, P),
$\forall SC_i \in Cvp_2$, si $SC \mathbf{p} vp_2 C_2, \Rightarrow \partial vp_2(SC_i) < \partial vp_2(C_2)$.
Type d'une sous-classe est inférieur au type de sa super-classe. Ceci est vrai à une union près. Donc :
$\partial vp_2(SC_i) + Att(P) < \partial vp_2(C_2) + Att(P)$

1.1.3. Expression des accès inter-points de vue en ODL

Les accès inter-points de vue sont définis dans les classes des schémas points de vue consommateurs. Tout attribut de type littéral peut être partagé par un autre schéma point de vue. Nous ne considérons pas le partage des relations entre entités. Les

relations ne sont donc pas partagées. La définition d'un schéma point de vue est enrichie par les données des différentes passerelles le liant aux autres schémas points de vue. Nous introduisons la clause "**access viewpoint**" en ODL pour exprimer une telle extension. La syntaxe de la définition des classes d'un schéma point de vue devient alors :

```
class <viewpoint_class_name> extend <referential_class_name>
  {
    viewpoint-part:
        {
        [<attribute_definition>]*
        [<relationship_definition>]*
        }
    referentiel-part:
        {
        [<attribute_definition>]*
        [<relationship_definition>]*
        }
    access-viewpoint_part <viewpoint_name>:
        {
        [<attribute_definition>]*
        }*
    }
```

La figure 2 présente un extrait de la définition du schéma point de vue *Comptable* en spécifiant ses accès inter-points de vue.

```
class  Part-externe_commercial extend Part-externe (extent part-externes_commercial key nom, telephone) {
      referentiel part :
            attribute string nom;
            attribute string telephone;
            attribute string adresse;
}

class client_commercial:Part-externe_commercial extend Client (extent clients_commercial key nom, telephone)
{
access-viewpoint_part Comptable:
     attribute integer encours_a_regler;
viewpoint part :
     attribute integer  montant_cmd;
     attribute date date_derniere_cmd;
 referentiel part :
     relationship                set<Produit_commercial>a_achete                inverse Produit_commercial::a_été_acheté_par;
}

class  fournisseur_commercial:Part-externe_commercial extend fournisseur (extent
                                                fournisseurs_commercial  key nom, telephone)
{
    viewpoint part :
      attribute string  contact_commercial;
   referentiel part :
      relationship set<Produit_commercial> fournit inverse Produit_ commercial::est_fourni-par;
}

class  Produit_ commercial extend Produit (extent produits_commercial)
{
   access-viewpoint_part Comptable:
      attribute integer chiffre_affaires;
 viewpoint part :
      attribute integer  nbre_unites_vendues;
      relationship set<Bon_de_livraison_commercial> est_contenu_dans inverse
         Bon_de_livraison_commercial  ::contient;
     relationship set<Bon_de_commande_commercial> est_contenu_dans inverse
         Bon_de_commande_commercial :: contient;
   referentiel part :
      attribute string designation;
      attribute integer prix_unitaire;

}
……..
```

Figure 2. Exemple d'accès inter-points de vue

La spécification du partage des données dans les différents schémas d'une base de données multi-points de vue étant réalisé, qu'on est-il du partage de valeurs entre les objets, instances de classes de ces schémas?. Dans la section suivante, nous exposons les problèmes rencontrés pour un tel partage et la solution adoptée.

1.2. Le partage des données entre les objets

Pour mieux caractériser les mécanismes de partage de données entre objets dans le modèle MVDB, nous examinons de manière précise les différents types de partage persistant[2] des données qui peuvent être induits par une relation de partage.

1.2.1. Caractérisation des types de partage

Dans [Bardou et Dammy 'Bardou p43], trois formes de partage sont présentées : le *partage de noms*, le *partage de propriétés* et le *partage de valeurs*[3]. Une définition informelle de chaque type est donnée ci-dessous. Soit deux objets O_1 et O_2 et une relation de partage qui lie O_1 à O_2, c'est à dire qui fait partager les propriétés de O_2 par O_1.

Partage de noms: c'est la caractérisation la plus faible d'un partage persistant et concerne l'existence de propriétés. Les objets partagent donc le fait d'avoir une propriété de nom donnée.

Partage de propriété: c'est la plus forte caractérisation des trois formes de partage. Il concerne le partage des propriétés elles mêmes. Il intervient lorsque un nom donné identifie exactement la même propriété dans deux objets différents. En d'autres termes, si O_1 partage les propriétés de O_2, alors toute propriété de O_2 dont la valeur n'est pas localement définie dans O_1, est également une propriété de O_1.

Partage de valeurs: dans cette forme de partage, ce sont les valeurs de propriétés qui sont partagées. Autrement dit, si l'objet O_1 partage les valeurs des propriétés de l'objet O_2 alors toute propriété de O_1 dont la valeur n'est pas localement définie a la même valeur que la propriété de O_2 de même nom.

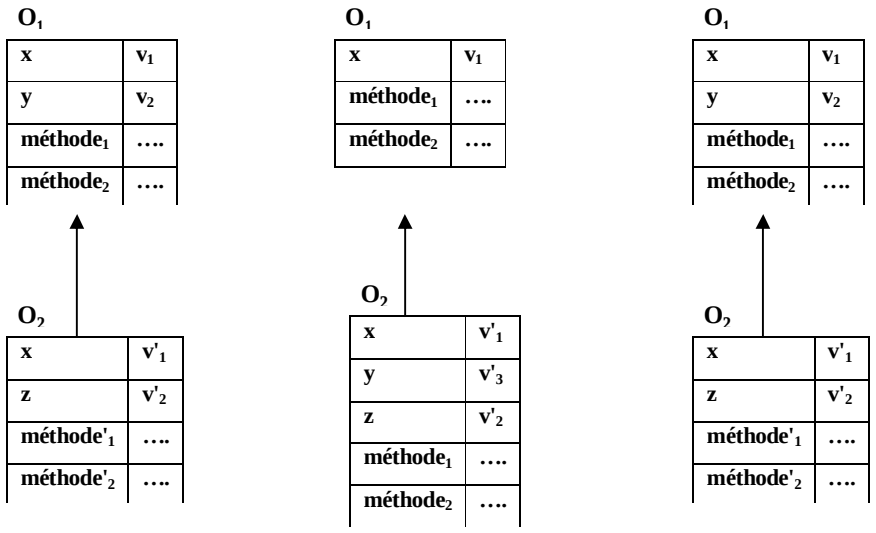
D'après les définitions précédentes, on conclut que les deux formes de partage de valeurs et de propriétés impliquent le partage de nom. Cependant, si le partage de noms est facilement saisi, il n'en est pas de même pour les deux autres formes de partage dont la différence est très peu distinguée. En fait, le partage de valeurs peut être considéré comme une forme restreinte de partage de propriétés. La différence essentielle entre eux est la suivante: le partage de valeurs n'impose qu'une égalité de valeur pour les propriétés non redéfinies dans O_1 alors que le partage de propriétés impose l'identité de celles-ci. Cette nuance prend tout son sens lorsque l'on considère la modification de telles propriétés.

Pour l'exemple de la figure 3, O_1 et O_2 partagent la valeur de deux variables V_1 et V_2.

---

[2] Le qualificatif "Persistant" se rapporte à la durée pendant laquelle l'utilisateur est assuré que le partage est effectif. Contrairement au partage "Ponctuel", le partage persistant suppose que tant qu'un concept admet une relation de partage vers un autre, il partage les propriétés de ce dernier.

[3] Ces appellations ont été inspirées par celles d'héritage de noms et d'héritage de valeurs de [Duccurneau 'Bardou p43]



(a) *: avant la modification du y pour O_2*

(b) *: après la modification du y avec du partage de valeurs*

(c) *: après la modification du y avec du partage de propriétés*

Figure 3. Les types de partage de données

Soit à modifier la valeur de V_2 dans O_2. Une réalisation de cette modification avec du partage de valeurs conduit à une redéfinition de V_2 par la valeur V'_3 au niveau de O_2 (voir figure 3.b) On considère dans cette interprétation que O_1 et O_2 ont chacun sa propriété y et dénote, dans ce cas deux entités différentes. La valeur partagée doit être considérée comme la vraie valeur du parent O_1 et comme une valeur par défaut pour l'enfant O_2. La figure 3.(c) montre l'interprétation de la modification dans le cas d'un partage de propriétés. O_1et O_2 doivent partager la même valeur de la propriété V_1. On considère alors qu'il y'a une seule valeur de la propriété y pour les deux objets et la modification est effectuée de la même manière que si le message de modification a été envoyé à l'objet O_2. Les états des deux objets ne sont pas distincts pour la valeur de y, puisque toute modification du parent O_1 peut modifier l'enfant O_2 et réciproquement. Les deux objets dénotent donc la même entité du domaine. O_1 est une représentation de cette entité et O_2 est une extension de O_1. Cependant, dans le contexte de notre travail, nous nous intéressons tout particulièrement à ces deux types de partage qui se rapprochent clairement de notre problématique pour le partage de valeurs d'attributs des objets multi-points de vue.

1.2.2. Quel type pour le partage des valeurs d'attributs entre les objets multi-points de vue?

Nous rappelons que dans le modèle MVDB, la représentation multi-points de vue suppose que plusieurs objets sont utilisés pour la représentation d'une seule entité du domaine. L'aspect général de cette entité est représenté dans un objet de base auquel sont reliés les autres objets qui représentent des spécialisations de cet aspect ou encore des extensions de l'objet de base selon différents points de vue. Chaque objet, qu'il soit de base ou une extension, possède un état qui lui est propre. Etant donné que ces états concernent une et une seule entité, il importe de partager entre ces objets les valeurs de

certains attributs. Cependant, tout partage de données suppose l'existence d'un lien entre ces objets. Néanmoins dans le modèle de classes sur lequel se base les concepts du modèle MVDB, il n'existe aucun lien entre objets et par conséquent aucun mécanisme de partage entre les instances d'une même classe et encore moins entre les instances de classes différentes. En effet, les deux uniques types de partage dans un modèle de classes sont :

– L'instanciation qui permet le partage des variables et des méthodes (nom et code) définies dans une classe par toutes ses instances.

– L'héritage qui permet le partage des variables et des méthodes (nom et code) des super-classes par les sous-classes.

L'instanciation peut être assortie d'une procédure d'initialisation; deux instances d'une même classe nouvellement créées ont la même structure et leurs variables sont initialisées avec les mêmes valeurs. Ceci est induit par le fait que l'héritage des méthodes dans un modèle de classes est réalisé par un partage de propriétés. Néanmoins, les valeurs des variables d'instances sont modifiées de façon complètement indépendante. La seule relation qui lie deux instances d'une même classe est qu'elles sont issues de la même classe et ne partagent que les noms de leurs propriétés.

Cette limitation du modèle de classes nous a conduit à la définition de deux liens entre objets définis formellement au §2.2.3. du chapitre 3 et au §1.1. du présent chapitre, respectivement:

- le lien *VP-Extension* qui relie un objet point de vue à son objet de base. Ce lien est réalisé à l'aide du concept de référent global et référent local.
- le lien *VP-Dependency* qui relie un objet point de vue à un autre objet point de vue et qui est réalisé via le concept de passerelle.

La question qui se pose à ce niveau est par quel type de partage doit-on caractériser les relations entre les objets multi-points de vue? Et plus précisément doit-on considérer du partage de valeurs ou du partage de propriétés?.

Dans la figure 4, nous considérons un client C1, instance de la classe Client du référentiel de notre application GEF-DP. C1 est décrit selon le point de vue *Comptable* et le point de vue *Direction*. Les objets points de vue auxquels C1 est lié par *VP-Extension* sont C11 et C12, respectivement. En plus, C11 et C12 sont liés par *VP-Dependency* selon la passerelle présentée en figure 1.

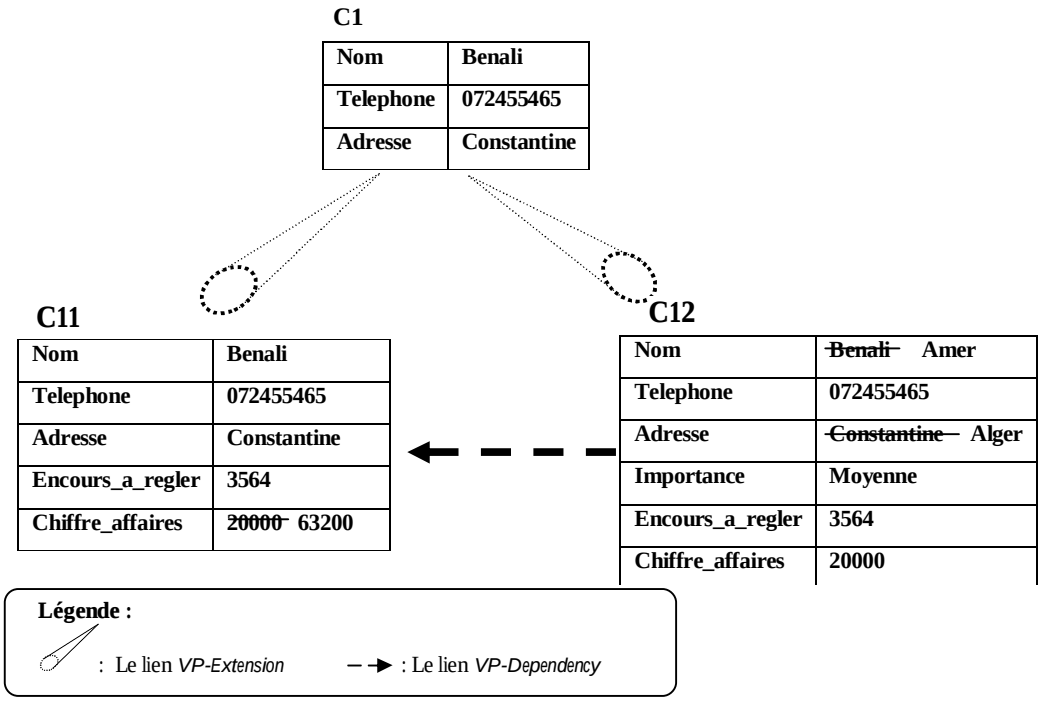


Figure 3. Le partage de données dans un objet multi-points de vue

Trois états sont alors associés à l'objet multi-points de vue C1. Un état de base et deux autres états points de vue. Soit à modifier le nom et l'adresse de C12. Un *partage de valeurs* conduit à une redéfinition des propriétés "Nom" et "Adresse" dans C12 en leur attribuant de nouvelles valeurs. Cette redéfinition conduit à une incohérence globale de la description de l'objet C1. Le même problème se présente si on tente de modifier le chiffre_affaires dans C11.

En effet, les trois objets sont censés dénoter une seule et même personne du domaine, et le *partage de propriétés* est donc parfaitement justifié. Que l'on s'intéresse à C1 selon le point de vue *Comptable* ou le point de vue *Direction* ou simplement en tant que Client, C1 n'a qu'un seul nom et celui-ci doit rester le même, quel que soit le point de vue considéré. Tout message d'affectation pour modifier la valeur de la variable nom, peut être envoyé à n'importe lequel des trois objets. Cependant, cette modification est prise en compte au niveau de C1 et elle est immédiatement effective, en vertu du *partage de propriétés*, pour l'ensemble des trois objets, c'est-à-dire l'ensemble de la représentation de C1.

Malheureusement, le problème qui se pose est que ce partage est inexistant dans le modèle objets à base de classes sur lequel se base MVDB. Ce qui nous fait tourner vers les modèles à prototypes tels définis dans [liberman ‘hala p117] et qui sont dotés du mécanisme de *délégation* qui permet l’héritage de valeurs des propriétés entre les objets. Donc l’héritage dans un modèle de classes est défini entre classes, c'est-à-dire entre des descriptions d’objets, alors que dans un modèle à prototype avec délégation l'héritage intervient directement au niveau des objets. Nous nous intéressons particulièrement à cette différence dans le niveau d’abstraction auquel la relation

d'héritage intervient. Dans ce qui suit, nous présentons succinctement le mécanisme de délégation avant d'aborder son adaptation aux besoins de notre modèle.

1.2.3. La délégation

La délégation est un mécanisme d'héritage utilisé dans la plupart des langages à prototypes. Dans ces langages, la notion de classe est inexistante et le seul concept de base est l'objet. Un objet a sa structure propre et contient son comportement, les deux pouvant évoluer simplement. Autrement dit, l'objet n'est pas lié à une description générique quelconque. Un objet possède également des fonctionnalités qui lui permettent de générer d'autres objets appelés les objets "extension". Un objet extension ou un "enfant" partage toutes les caractéristiques de son objet géniteur appelé son "prototype" ou son "parent". Le mécanisme de délégation est le seul mécanisme qui permet de réaliser le partage persistant entre les objets, c'est-à-dire qu'il permet de lier deux objets aussi longtemps qu'il existe un lien de délégation de l'un à l'autre. Cependant, le partage par délégation peut être caractérisé comme étant du *partage de valeurs* ou encore du *partage de propriétés*, en fonction de la manière dont est résolue l'affectation des variables en présence de la délégation [BD96, Bardou article P5]. Par exemple, dans le système GARNET [MGD 90, Bardou p5], le partage réalisé par la délégation est du *partage de valeurs* alors que dans le langage SELF [US87, Bardou p5] c'est du *partage de propriétés* qui est réalisé par la délégation.

1.2.4. Adaptation de la délégation dans le modèle MVDB

Dans la section précédente, nous avons spécifié la sémantique du partage induite par les différents liens qu'un objet multi-points de vue peut établir avec d'autres objets. Cette sémantique se résume en deux points : **le partage est persistant et il est de type partage de propriétés.**

Cette sémantique permet de maintenir la cohérence entre les objets comme représentation d'une même entité du domaine. En plus, les propriétés partagées ne sont modifiées que par les objets ou elles ont été initialement définies. Ce qui permet d'assurer la protection de ces données.

Le type de partage préconisé par notre modèle est une possibilité caractéristique de la délégation, mécanisme absent des modèles de classes. La question fondamentale qui se pose alors est comment adapter la délégation dans le modèle MVDB à base de classes?. Plus précisément comment adapter le mécanisme de partage pour réaliser la délégation?. Pour cela nous procédons à une *instanciation partielle* et une *délégation explicite* en ce qui concerne les objets points de vue.

L'*instanciation partielle* consiste à appliquer le mécanisme d'instanciation du modèle de classes uniquement sur les attributs propres à la classe du schéma point de vue, classe d'appartenance de l'objet (ou l'instance) ou ses super-classes. Par contre, les valeurs des attributs partagés, c'est-à-dire les attributs issus du schéma référentiel ou des autres schémas points de vue sont obtenus par *délégation explicite*. La *délégation explicite* consiste en un envoi de messages explicites aux objets concernés pour l'obtention des valeurs des propriétés partagées. Donc, un objet point de vue délègue explicitement l'accès à la valeur d'un attribut partagé à un autre objet (de base ou objet point de vue). Cependant, la mise en œuvre de ce procédé exige une technique et un

effort de programmation non négligeable. Nous y reviendrons dans la mise en œuvre au chapitre 5.

2. La cohérence des données dans MVDB

Dans cette section, nous nous proposons d'étudier la cohérence des données dans le modèle MVDB. Un état de la base de données est cohérent s'il vérifie les critères de cohérence définis sur la base de données. Ces critères s'expriment par des contraintes d'intégrité. Celles-ci sont exhibées lors de la conception de la base et leur validité est vérifiée sur les données lors de l'exploitation. S'assurer de la validité des contraintes d'intégrité permet donc, dans une certaine mesure, d'éviter que les données stockées soient aberrantes. Elles protègent par conséquent, en partie, des mises à jour erronées dues, par exemple, à l'inattention ou au manque d'information de l'utilisateur.

Dans les systèmes de bases de données, les contraintes d'intégrité sont traditionnellement exprimées dans le schéma global et sont vérifiées pour toutes les instances de la base de données, c'est-à-dire pour toutes les instances de l'extension de chaque classe. Cependant, la représentation multi-points de vue dans MVDB, où le schéma d'une base de données est structuré en plusieurs schémas points de vue, nous incite à proposer de prendre en compte les contraintes à deux niveaux :

- A un niveau local au sein des schémas.
- A un niveau global au sein du référentiel.

2.1. Types de contraintes étudiées

Les contraintes d'intégrité ont principalement été étudiées dans le cadre du modèle relationnel. Ensuite, l'intérêt s'est porté sur les modèles à objets (cf [Bauzer Medeiros91, Bouaziz95, Delannoy94, Morejon94, Nassif91]). Il s'agit, en général, d'adapter des résultats élaborés dans le cadre du modèle relationnel aux concepts des objets tel est le cas pour l'étude des vues objets ou encore du langage d'interrogation OQL.

Dans le cadre du modèle MVDB , nous classifions les contraintes d'intégrité en deux types : les contraintes intra-schéma et les contraintes inter-schémas. Ces contraintes sont exprimées au sein des classes et qui s'appliquent individuellement à leurs instances. En effet, nous nous situons dans le cadre du modèle "objet" qui prône la centralisation des descriptions au sein des classes et nous voulons respecter cette centralisation pour l'expression des contraintes.

2.1.1. Les contraintes intra-schéma

Les contraintes intra-schéma sont réparties sur les différents schémas associés à une base de données multi-points de vue: le schéma référentiel et les schémas points de vue. Elles permettent d'assurer la cohérence des objets dans une représentation (de base ou point de vue). Le besoin d'exprimer des contraintes d'intégrité intra-schéma est similaire à celui des contraintes définies dans les modèles à objets mono-points de vue. Dans ces derniers, il existe de nombreuses formes de contraintes d'intégrité (cf [Bauzer Medeiros91, Bouaziz95, Delannoy94, Morejon94, Nassif91]). Cependant, les contraintes, que l'on rencontre le plus fréquemment et auxquelles nous nous limitons, sont :

- les contraintes d'intégrité référentielle (CIR),
- les contraintes de cardinalité,
- les contraintes d'intégrité statiques sur les attributs.

Parmi les contraintes étudiées dans le cadre des bases de données à objets que nous n'abordons pas, citons les contraintes d'intégrité dynamiques (cf [Delannoy94]) dont l'étude nécessiterait l'introduction d'une logique temporelle et les contraintes relatives à la relation d'agrégation (cf [Bertino93-p36]).

2.1.1.1. Les contraintes référentielles

Les contraintes d'intégrité référentielles sont relatives aux attributs non littéraux mettant en relation les entités dans un schéma objet. Les relations de ODMG en sont des exemples type. Gérer une contrainte référentielle consiste, lors de la suppression d'une instance, à supprimer toutes les occurrences des relations ayant cette instance pour extrémité. Il ne peut ainsi exister de références vers des instances inexistantes dans la base de données. Dans le cas de violation d'une contrainte référentielle, un traitement compensatoire est appliqué afin de supprimer l'instance là où elle est référencée.

Prenons le cas de la relation du référentiel de notre application, "a_acheté/a_été_acheté_par " entre les entités "Client" et "Produit"(cf. Annexe A). Supposons qu'il existe un client "c" ayant acheté un produit "p". Supposons ensuite que cette instance "p" soit supprimée. L'occurrence entre "c" et "p" est alors également supprimée afin d'assurer l'intégrité référentielle.

2.1.1.2. Les contraintes de cardinalité

Les contraintes de cardinalité concernent les attributs de type ensemble ou liste. Elles permettent d'exprimer le nombre maximal ou minimal des éléments de cette liste ou cet ensemble que chaque instance de la classe doit vérifier. Par exemple, l'attribut "Fournit", dans la classe "Fournisseur" du référentiel détermine une cardinalité minimale de un. Ainsi, toute instance de fournisseur doit avoir au moins un élément "Produit" dans cette liste. Les contraintes de cardinalité sont exprimées par des prédicats du premier ordre.

2.1.1.3. Les contraintes d'intégrité statiques

Les contraintes d'intégrité statiques sont exprimées sur les attributs (un ou plusieurs) d'une classe dans un schéma objet en utilisant des prédicats.

Dans MVDB, pour respecter la localité à un point de vue, un prédicat de contraintes ne peut être exprimé que sur les propriétés pertinentes pour le schéma point de vue correspondant et non pas sur les propriétés partagées. Nous rappelons qu'un objet dans un point de vue partage les valeurs des attributs de son objet de base et celles des objets point de vue auxquels il est lié. Cependant, il n'existe aucun mécanisme d'expression de chemins pour modifier les valeurs partagées. Une contrainte exprimée sur un attribut partagé n'a donc pas de sens car aucun moyen pour un traitement compensatoire ne peut être appliqué. La requête est dite dans ce cas *invalide*.[4]

[4] Une contrainte est dite valide si et seulement si, en cas de violation, un traitement compensatoire peut être appliqué pour la rendre à nouveau vérifiée.

Prenons l'exemple de la l'extension de la classe "Produit" par le point de vue "Comptable". Dans la classe "Produit_comptable", on peut définir la contrainte "encours_a_regler < chiffre_affaires". Cette contrainte exprimée seulement sur les attributs définis dans ce point de vue est valide. Ainsi, une opération issue des clients du point de vue "Comptable", augmentant la valeur du "encours_a_regler" dans une instance de "Produit" et provoquant une violation de cette contrainte ne peut conduire au rejet de sa transaction englobante. Un traitement compensatoire au sein du point de vue "Coptable" visant à augmenter le chiffred'affaires du même produit doit être invoqué.

Un exemple de contrainte faisant intervenir une propriété partagée peut être exprimée dans la classe "Produit_grand_public_fabrivation", extension de la classe "Produit_grand_public" dans le point de vue "Fabrication". La contrainte "nbre_unites_vendues > 20000" est invalide. En effet, "nbre_unités_vendues" est un attribut partagé et les clients de ce point de vue n'ont aucun moyen de rendre un tel prédicat vérifié pour une instance de Produit dont la valeur du nombre des unités vendues a augmenté dans le point de vue "Commercial".

2.1.2. Les contraintes inter-schémas

Les contraintes inter-schémas sont définies uniquement sur la hiérarchie constituant le référentiel de la représentation multiple. Elles sont spécifiées dans les différentes classes du référentiel. Elles assurent une compatibilité entre les différentes représentations partielles des objets. Une contrainte inter-schéma est une contrainte globale qui exprime une relation qui lie des représentations différentes d'un objet multi-point de vue. Elle peut porter sur:

- la classe d'appartenance d'une représentation de l'objet multi-points de vue dans un point de vue,
- sur l'existence d'une représentation dans un point de vue,
- sur des attributs relatifs à plusieurs représentations de l'objet multi-points de vue. Ces attributs appartiennent à des structures de classes différentes.

L'intégrité sémantique d'un univers de discours représenté avec MVDB, est donc exprimée grâce à ces deux types de contraintes. Afin de spécifier ces contraintes, nous avons défini un langage d'expression prédicatif du premier ordre. Il est basé sur les notions simples de termes et de formules. Il s'agit d'une approche déclarative dans la mesure où nous ne nous intéressons pas au maintien de ces contraintes, mais seulement à leur expression. Dans ce qui suit, nous présentons le langage de contraintes de MVDB.

2.2. Le langage d'expression des contraintes de MVDB

Les contraintes d'intégrité sont habituellement exprimées déclarativement. Spécifier les contraintes déclarativement au lieu d'intégrer les vérifications appropriées dans les programmes présente de nombreux avantages :

- le concepteur de la base peut se concentrer sur la détermination des contraintes plutôt que sur leur vérification.

- les programmes applicatifs sont plus simples et plus modulaires grâce à la factorisation des aspects relatifs à la vérification des contraintes. Les mêmes tests n'ont plus à être écrits dans les différents programmes utilisant la base.

- on est sûr que les mêmes contraintes sont vérifiées dans tous les programmes, même s'ils n'ont pas été écrits par le même programmeur.

- il est possible d'automatiser la vérification des contraintes, à l'instar des langages d'interrogation déclaratifs qui ont permis l'automatisation de l'optimisation de l'évaluation des requêtes[5].

- la modification des contraintes ne nécessite pas de modifier les programmes applicatifs. Cet argument, souvent cité.

Pour exprimer les contraintes dans le modèle MVDB, nous proposons un langage d'expression basé sur la logique du premier ordre. L'expression déclarative des contraintes d'intégrité en utilisant la logique du premier ordre sans symbole de fonctions est apparue avec le modèle relationnel.

Le langage d'expression des contraintes de MVDB est restreint aux constantes, aux variables, aux fonctions et aux prédicats. Il ne contient pas de quantificateurs existentiels et universels. Ce langage permet de générer des formules bien formées construites à l'aide d'un alphabet constitué des symboles suivants :

- un ensemble de constitué des valeurs;
- un ensemble de variables constitué des noms des attributs, des noms des points de vue et des noms des classes;[2]
- un ensemble de symboles de fonctions avec arité. Cet ensemble est constitué des symboles : ". , -> , card , element" et les fonctions arithmétiques usuelles : "+ , - , * , / ";
- un ensemble de symboles de prédicats avec arité. Cet ensemble contient les prédicats usuels de comparaison entre les valeurs atomiques : "> , < , = , ≠ , ≤ , ≥".
- Les connecteurs logiques.

2.2.1. Les fonctions du langage de contraintes

Pour le langage des contraintes de MVDB nous définissons les fonctions suivantes :

– La fonction **card** rend le nombre d'éléments d'un attribut de type liste d'une classe c. Cette fonction est définie comme suit :

card : att*(c) **à** N.

Att*(c) est une fonction qui rend l'ensemble des attributs qui sont définis dans la classe cainsi que ceux hérités. La précondition de cette fonction est qu'elle ne s'applique que sur des attributs de type liste.

[5] Remarquons que contraintes d'intégrité et langages de requêtes sont deux concepts relativement proches : une contrainte d'intégrité peut être vue comme une requête devant rendre un résultat vide.

– La fonction **element** permet d'accéder à un élément d'une liste. Cet élément est repéré grâce à son rang dans la liste (son rang est un entier compris entre 1 et la cardinalité de la liste). Cette fonction est définie comme suit :

element : att* (c), N **à** V.

Cette fonction prend en paramètre un attribut et un indice entier, et rend une valeur. La précondition est que l'attribut pris en paramètre soit de type liste. Si la fonction prend en paramètre un indice i alors ce dernier doit être compris entre l et la cardinalité de l'attribut.

– La fonction "**.**" permet l'accès à la valeur d'un attribut d'un objet instance directe ou indirecte d'une classe. Elle est définie comme suit :

. : att* (c) **à** Val

Cette fonction prend en paramètre un objet instance directe ou indirecte d'une classe c et un attribut de c. Elle rend comme résultat une valeur. Rappelons qu'une valeur peut être une valeur atomique, une valeur structurée, un ensemble, une liste de valeurs ou un objet (cf. Définition 1. du chapitre 3).

.(o, a) = v $\Rightarrow$ v$\in$ Domain(Type (a))

– La fonction "->" permet d'accéder à un objet point de vue qui représente une entité réelle selon un point de vue donné. La précondition de cette fonction est que l'objet point de vue existe. Elle est définie comme suit :

-> : π * (c) , VP **à** Ovp

Cette fonction prend en paramètre un objet instance directe ou indirecte d'une classe et un point de vue. Elle rend un objet point de vue. Selon que l'objet pris en paramètre est un objet multi-point de vue ou un objet point de vue relatif à un point de vue, on a :

-> (mvo, vp) = vo' $\Leftrightarrow$ vp-extension(vo', mvo)

-> (vo, vp) = vo' $\Leftrightarrow \exists$ mvo, vp-extension(vo, mvo) et vp-extension(vo', mvo)

mvo est un objet multi-point de vue et vo est un objet point de vue.

2.2.2. Les prédicats du langage

Pour le langage des contraintes de MVDB nous définissons les prédicats suivants :

– Le prédicat "**exist**" permet de tester si une entité réelle possède un objet point de vue qui la décrit dans un point de vue particulier. Il est défini comme suit :

exist : i*(c), VP

Ce prédicat prend en paramètre un objet instance directe ou indirecte d'une classe. Selon que l'objet soit un objet multi-point de vue (mvo) ou un objet point de vue (vo) relatif à un point de vue vp, on a :

exist(mvo, vp) ⇔ (∃ vo, vp-extension(vo, mvo))

exist(mvo, vp) ⇔ ∃ vo', mvo, vp-extension(vo, mvo) et vp-extension(vo', mvo)

– Le prédicat "**test**" permet de tester si un objet est instance d'une classe. Il est défini comme suit :

test : O, C **à** Bool

Ce prédicat prend en paramètre un objet et une classe. On a : test(o, c) ⇔ (o∈ π*(c)

2.2.3. Les primitives de base du langage des contraintes de MVDB

Dans cette, nous présentons les primitives suivantes du langage d'expressions des contraintes d'intégrité qui représentent les termes et les formules définies sur les classes du référentiel et des classes des schémas points de vue.

Définition 4. Soit Termes(c) l'ensemble des termes d'une classe c. Cet ensemble est défini récursivement par :

1. Toute valeur de v∈ Val appartient à Termes(c),
2. Les noms des attributs qui sont définis dans c ainsi que ceux hérités sont des termes de c; att*(c) ⊂ Termes(c),
3. Si f est un symbole de fonction d'arité n (f ∈ {+, -, *, /, element, card, .} et si t1,…tn sont des termes de la classe c sur lesquels la fonction f peut être appliquée, on a alors : f(t1,…tn) ∈ Termes(c),

Définition 5. Soit Form-Atomic(c) l'ensemble des formules atomiques d'une classe c. Cet ensemble est défini comme suit:

Si p est un symbole de prédicat p ∈{<, >, ≥, ≤, ≠, =} et si t1 et t2 sont des termes de c sur lesquels le prédicat p peut être appliqué, alors on a : t1 p t2 ∈ Form-Atomic (c).

Définition 6. Soit Formules(c) l'ensemble des formules bien formées qui peuvent être générées à partir d'une classe c. Cet ensemble est défini de la manière suivante :

- ∀ fo ∈ Atomic-fo(c) : fo ∈ formules(c)
- ∀ fo ∈ Atomic-fo(c) : (fo) ∈ formules(c)
- ∀ fo ∈ Atomic-fo(c) : ┐(fo) ∈ formules(c)
- ∀ fo1, fo2 ∈ Atomic-fo(c) : (fo1) ⇒ (fo2) ∈ formules(c)
- ∀ fo1, fo2 ∈ Atomic-fo(c) : (fo1) Λ (fo2) ∈ formules(c)

Exemples de formules :

- Designation : Attribut de la cvlasse Produit.

- Card(a_ete_achete) : Cardinalité de la liste des clients qui ont acheté un produit.
- Prix_unitaire>0 : le prix unitaire d'un produit doit être supérieur à 0.
- chiffre_affaires <= nbr_unites_vendues*prix_tarif : dans le point de vue comptable, le chiffre d'affaires d'un produit doit être supérieur au produit du nombre des unités vendues du produit et son prix-tarif.

Définition 6. Soit Vp-Termes(mvc) l'ensemble des vp-termes d'une classe du référentiel. Cet ensemble est défini comme suit :

1. Si vp est le nom d'un point de vue, les termes de la classe multi-points de vue mvc ainsi que ceux des classes point de vue associées à la classe mvc sont des vp-termes de mvc :

   Termes(mvc) $\cup$ ($\cup_{vc,\ vp\text{-}extension}$ (vc, mvc) Termes (vc))) $\subset$ VP-Termes(mvc)
2. Si vp est le nom d'un point de vue, on a : ->vp $\in$ VP-Termes(mvc)
3. si vpt1 est un vp-terme de la classe mvc, et vp est le nom d'un point de vue, on a : vpt1 -> vp $\in$ VP-Termes(mvc)
4. Si f est un symbole de fonction d'arité n (f $\in$ {+, -, *, /, element, card, .} et si t1,…tn sont des vp-termes de la classe c sur lesquels la fonction f peut être appliquée, on a alors : f(t1,…tn) $\in$ VP-Termes(c),

Définition 7. Soit VP-Form-Atomic l'ensemble des vp-formules atomiques d'une classe multi-points de vue mvc. Cet ensemble est défini comme suit :

1. Si vp est le nom d'un point de vue, alors : exist(vp) $\in$ VP-Form-Atomic (mvc)
2. Si vpt est le nom local d'une classe point de vue, alors exist(vpt) $\in$ VP-Form-Atomic (mvc)
3. Si vc est le nom local d'une classe point de vue, alors : exist(vc) $\in$ VP-Form-Atomic(mvc).
4. Si p est un symbole de prédicat p $\in$ {<, >, $\geq$, $\leq$, $\neq$, =} et si vpt1 et vpt2 sont des vp-termes de c sur lesquels le prédicat p peut être appliqué, alors on a :
   vpt1 p vpt2 $\in$ VP-Form-Atomic(mvc).

Définition 6. Soit VP-Form(mvc) l'ensempble des vp-formules d'une clases multi-points de vue. Cet ensemble est défini comme suit :

1. VP-Form-Atomic(mvc)$\subset$ VP-Form(mvc)
2. $\forall$ fo $\in$ VP-Form-Atomic(mvc) : fo $\in$ VP-Form(mvc)
3. $\forall$ fo $\in$ VP-Form-Atomic(mvc) : $\neg$ (fo) $\in$ VP-Form(mvc)
4. $\forall$ fo1, fo2 $\in$ VP-Form-Atomic(mvc): (fo1) $\Rightarrow$ (fo2) $\in$ VP-Form(mvc)
5. $\forall$ fo1, fo2 $\in$ VP-Form(mvc) : (fo1) $\Rightarrow$ (fo2) $\in$ VP-Form(mvc)

6. $\forall$ fo1, fo2 $\in$ VP-Form(mvc): (fo1) Λ (fo2) $\in$ VP-Form(mvc)

Exemples de VP formules :

- (->Commercial) exist (Client_commercial) $\Rightarrow$ (->Comptable) exist (Client_comptable) : toute représentation d'un client dans le point de vue "Commercial" engendre obligatoirement sa représentation dans le point de vue "Comptable".
- (->fabrication) exist (Client_fabrication) $\Rightarrow$ (->Comptable) exist (Client_comptable) : toute représentation d'un client dans le point de vue "Fabrication" engendre obligatoirement sa représentation dans le point de vue "Comptable".
- (->Commercial) exist (Produit_grand_public_commercial) $\wedge$ (nbre_unites_vendues < 20000) $\Rightarrow$ $\neg$(->fabrication) exist (Produit_grand_public_fabrication) : tout produit grand_public représenté dans le point de vue "Commercial" et ayant un nombre d'unités vendues inférieur à 20000 ne peut pas être fabriqué (il n'a pas de représentation du point de vue "Fabrication")
- (->Commercial) exist (Produit_professionnel_commercial) $\wedge$ (nbre_unites_vendues < 300) $\Rightarrow$ $\neg$(->fabrication) exist (Produit_professionnel_fabrication) : tout produit professionnel représenté dans le point de vue "Commercial" et ayant un nombre d'unités vendues inférieur à 300 ne peut pas être fabriqué (il n'a pas de représentation du point de vue "Fabrication")

3. Conclusion

Dans ce chapitre, nous avons considéré les mécanismes de protection, de partage et de cohérence des données dans le modèle MVDB. Une typologie des contraintes a été spécifiée. Celle-ci classifie les contraintes en deux familles: les contraintes intra-schémas qui assurent une cohérence des objets localement à chaque schéma dans la base multi-points de vue et les contraintes inter-schémas qui assurent une compatibilité entre les différents schémas. Un langage d'expression de ces contraintes est proposé. Cependant, la spécification d'une méthode de calcul et de maintien de la cohérence reste une perspective imminente pour notre travail.

Chapitre 5

Où l'on présentera des aspects d'implémentation du modèle MVDB présenté aux chapitres précédents. Cette mise en œuvre est réalisée avec le langage JAVA. JAVA est un langage objets simple, sûr et idéal pour réaliser des applications distribuées robustes. Nous l'utilisons comme un langage de spécification des schémas de bases de données multi-points de vue et comme un langage de programmation.

La mise en œuvre a donné lieu à un prototype appelé JAVA-MVDB constitué principalement de:

- *Des outils de conception et de génération de schémas.*
- *Des outils de création des objets.*

Cependant, nous rappelons que la construction des schémas dans MVDB se fait d'une manière distribuée. De ce fait, l'échange de données entre les différents sites est réalisé via des documents XML. On reconnaît pour ce langage de balisage type HTML son pouvoir de représentation lisible, simple, flexible et précise des données ainsi que son intégrabilité et son extensibilité. XML a été très vite promu dans la communauté informatique comme un format universel pour l'échange des données entre des applications distribuées.

Dans la première section du présent chapitre, nous présentons la démarche de conception d'une base de données multi-points de vue en MVDB. Celle-ci étant inspirée de la méthode VBOOM pour le développement des systèmes complexes basé sur le concept de point de vue. Ensuite, dans la section 2, nous introduisons le prototype JAVA-MVDB. Nous décrivons les différents outils de conception qu'il offre ainsi que le détail de génération automatique et distribuée d'un schéma de bases de données décrite selon des points de vue différents.

La deuxième partie de ce chapitre est consacrée aux extensions que nous proposons pour notre approche. Ainsi, nous considérons :

- *Une architecture distribuée basée sur la fédération pour le support et l'exploitation à bon escient des bases de données multi-points de vue.*
- *Une méthode d'intégration d'un nouveau composant déjà existant à une base de données multi-points de vue en cours de conception.*

1. Démarche de conception d'une base de données multi-points de vue

Le processus de développement avec MVDB n'étant pas l'objectif principal de cette thèse, nous ne présentons qu'une version simplifiée de la démarche associée à MVDB pour la conception d'une base de données multi-points de vue. Les phases principales de cette démarche sont inspirées de la méthode VBOOM [51] utilisés également dans le processus de développement avec VUML [66].

La conception d'une base de données dans MVDB s'effectue en 3 phases (cf. Figure 1): phase centralisée de construction du schéma référentiel, phase décentralisée de création de schémas point de vue, phase centralisée de fusion et de spécification des contraintes globales.

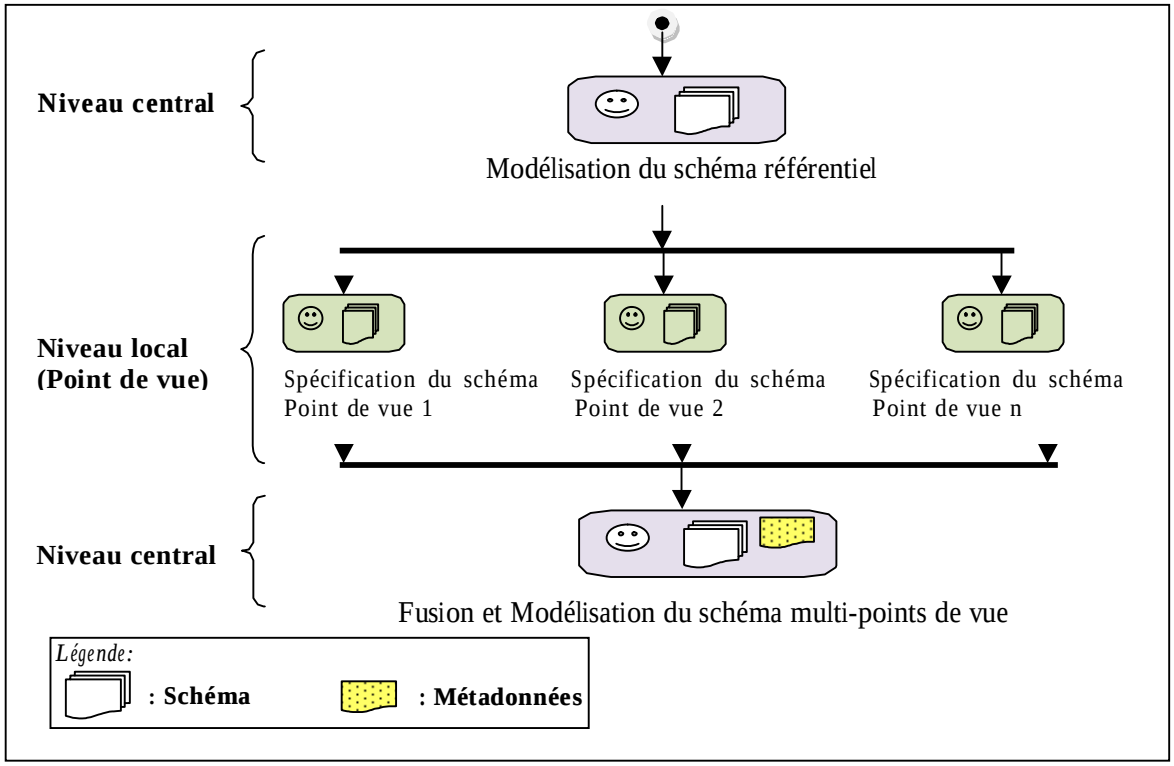


Figure 1. Démarche de conception d'un schéma multi-points de vue

1.1. Modélisation du schéma Référentiel (Niveau central)

La phase de conception du schéma référentiel permet de modéliser les exigences globales de l'application en termes de description globale des entités réelles du domaine. Cette phase est principalement réalisée par l'administrateur de la base de données, que nous appelons le coordinateur, avec intervention éventuelle des différents acteurs qui participent à la conception de cette base.

1.2. Spécification des schémas points de vue (Niveau local ou point de vue)

La deuxième phase consiste à spécifier des schémas par point de vue, c'est-à-dire par acteur. Ces schémas sont générés dans des sites différents à partir du schéma élaboré dans la première phase (cf. Figure 2). Le résultat de cette phase est un ensemble de schémas qui montrent clairement les informations pertinentes de chaque acteur participant à la conception.

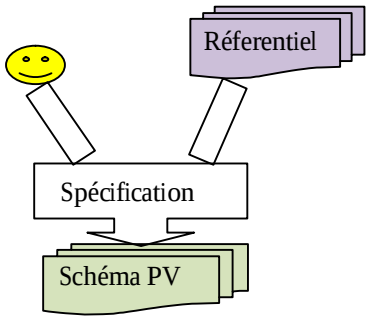


Figure 2. Spécification d'un schéma point de vue

1.3. Fusion et Modélisation du schéma multi-points de vue (Niveau central)

La troisième étape concerne la fusion des différents schémas et la mise à jour de la métabase (présentée ci-dessous) pour assurer la gestion globale du schéma multi-points de vue. Cette phase se fait d'une manière manuelle par le coordinateur de la base de données. Ce dernier doit procéder par une comparaison des schémas points de vue qui peut révéler des ambiguïtés ou des incohérences sur les classes (nommage, attributs, méthodes).

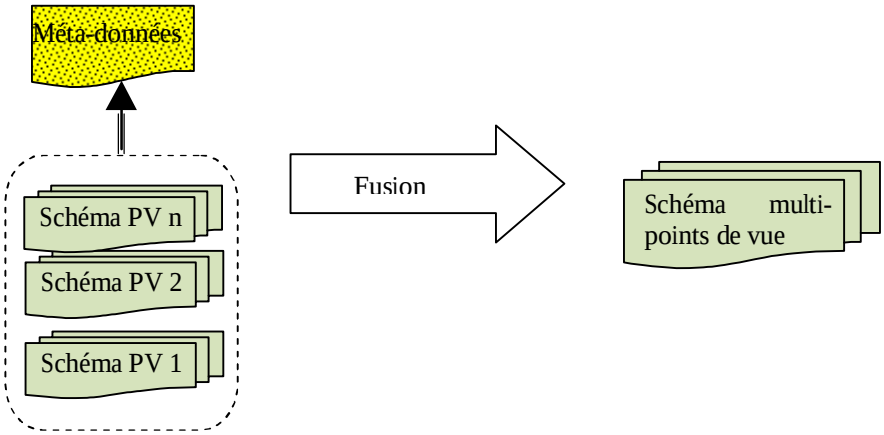


Figure 3. Fusion des schémas points de vue

Une fois le schéma de la base de données élaboré, il faut décrire les éventuelles dépendances entre les points de vue. Rappelons que dans MVDB, ces dépendances sont modélisées par des relations de dépendance «viewDependency». Ces relations seront

exploitées lors de la phase d'implémentation pour gérer la cohérence entre les points de vue. Aussi, des contraintes globales spécifiques à la gestion de la représentation des objets par points de vue sont exprimées dans une matabase.

2. Le prototype JAVA-MVDB

Dans cette section nous présentons le prototype JAVA-MVDB qui implantent les concepts du modèle MVDB présentés aux chapitres 3 et 4.

2.1. Description générale du prototype

Le prototype JAVA-MVDB permet la conception distribuée de bases de données orientées objets décrites selon des points de vue différents. Il est composé de:

- Outils de conception et de génération de schémas constitués de :
  1. Un browser central pour la conception du schéma référentiel. Cette conception est effectuée par un administrateur de la base de données multi-points et que nous appelons ici le *coordinateur*.
  2. Un browser distribué au niveau de différents sites pour permettre aux *spécialistes* de concevoir les schémas points de vue.
- Outils de création des objets constitués de :
  1. Un browser central qui permet au *coordinateur* d'instancier les classes du schéma référentiel, i.e. la création des objets multi-points de vue.
  2. Un browser distribué au niveau de différents sites pour permettre aux *spécialistes* d'instancier les classes des schémas points de vue en enrichissant la description de base des objets du domaine par des descriptions partielles selon des points de vue.

Le lanceur du prototype est une fenêtre principale (une capture d'écran est donnée en Figure 4) pour ouvrir une session de travail qui peut être soit pour le coordinateur ou pour un spécialiste. Selon le cas, les browsers de conception de schémas et de création des objets seront disponibles pour utilisation.

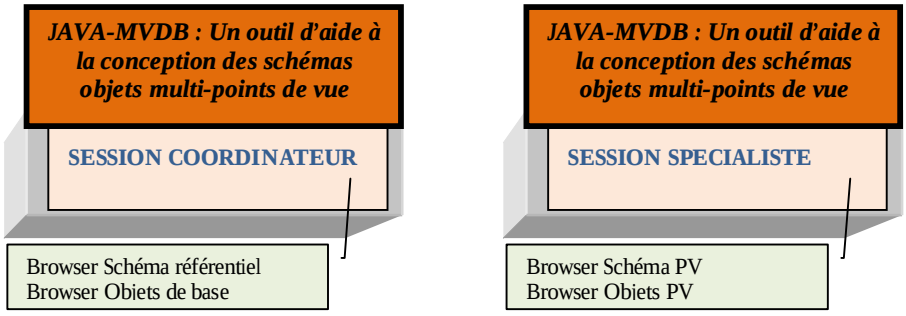


Figure 4. Les lanceurs du prototype JAVA-MVDB

2.2. Choix des outils d'implantation

Pour la mise en œuvre du prototype, nous avons choisi le couple JAVA-XML. En effet, L'utilisation ensemble de Java et XML est facilitée par le fait qu'ils ont plusieurs points communs tels que:

- L'indépendance de toute plateforme
- Ils sont conçus pour être utilisé sur un réseau

JAVA offre un mécanisme puissant qui permet la persistance des objets: la *sérialisation*. La sérialisation permet de mettre un objet sous une forme sous laquelle il pourra être reconstitué à l'identique. Ainsi il pourra être stocké sur un disque dur ou transmis au travers d'un réseau pour le créer dans une autre MVJ (Machine virtuelle JAVA). C'est le procédé qui est utilisé par CORBA et RMI. La sérialisation est aussi utilisée par les beans pour sauvegarder leurs états.

Au travers de ce mécanisme, Java fourni une façon facile, transparente et standard de réaliser cette opération. Il est de ce fait inutile de créer un format particulier pour sauvegarder et relire un objet. Le format utilisé est indépendant du système d'exploitation. Ainsi, un objet sérialisé sur un système peut être réutilisé par un autre système pour récréer l'objet. Cet aspect est utile dans une application distribuée comme la notre.

2.3. Browsers de schéma

Ce sont des browsers de conception des schémas multi-points de vue qui prennent en compte les aspects de représentation et de structuration des classes du domaine présentés aux chapitres 3 et 4. Chacun des différents browsers peut être activé en mode consultation ou en mode modification. Par exemple, si le concepteur est le coordinateur de l'application, les browsers du schéma référentiel et de la base référentiel seront activés en mode modification et les autres browsers des différents schémas points de vue et bases points de vue seront activés en mode consultation uniquement.

Dans ce qui suit nous décrivons le détail de ces outils, une capture d'écran est donnée pour chacun d'eux dans le cadre de l'exemple de l'annexe A.

2.3.1. Browser de conception du schéma référentiel

Le browser du schéma référentiel, présenté en figure 5, se compose de trois zones. Une zone d'édition placée à gauche et se compose d'une partie supérieure qui contient la liste des points de vue définie pour l'application. La sélection d'un point de vue affiche dans la partie en dessous la hiérarchie des classes du schéma point de vue correspondant. Le schéma référentiel en cours de création est affiché également dans cette zone. Un simple clic sur le nom d'une classe de cette hiérarchie permet de visualiser, dans une troisième partie en bas, le code JAVA généré automatiquement à la création de celle-ci.

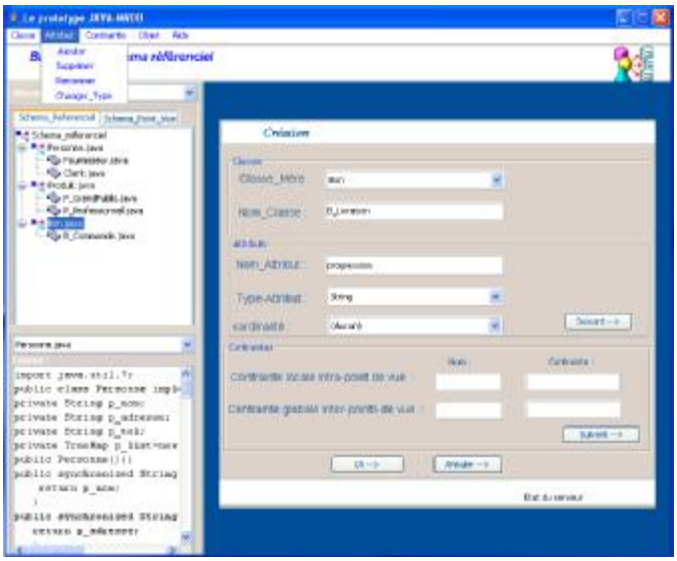


Figure 5. Le browser du schéma référentiel

Notons que les fichiers code JAVA des schémas points de vue sont détenus par leurs sites et ne peuvent être consultés encore moins modifiés au niveau de ce browser. L'autonomie de conception dans les sites est donc préservée. A la zone principale de création du schéma référentiel est associé un menu répertorié en opérations sur les classes (créer, supprimer, renommer,..), opérations sur les attributs des classes (ajouter, supprimer, renommer, changer type,..) et opérations sur les contraintes (définir, supprimer, modifier). Dans la figure 5, nous présentons l'opération de création d'une classe. La fenêtre de création est divisée en trois parties: une zone pour identifier la classe (nom et éventuellement sa super-classe), une zone pour la saisie des attributs de la classe. Un attribut peut être de type littéral ou de type référencé (une association dans le modèle ODMG). Dans ce dernier cas, une cardinalité est spécifiée. La troisième zone est consacrée à la spécification des contraintes d'intégrité (contraintes locales et contraintes globales).

2.3.2. Browser de conception du schéma point de vue

Un schéma point de vue correspond à une description partielle des entités du domaine selon un point de vue donné et à la spécification des passerelles réalisées par des accès inter-points de vue. L'interface de ce browser, présenté en figure 6, ne diffère pas de celle du browser du schéma référentiel en ce qui concerne la zone d'édition sauf que le code JAVA affiché est celui des classes du schéma point de vue en cours de création. Dans la zone de saisie, et pour la création d'une classe, une liste déroulante des classes du schéma référentiel est ajoutée pour éventuellement sélectionner la classe du référentiel qui sera étendue par la classe en cours de création. En plus, une zone des accès inter-points de vue est réservée pour sélectionner les attributs des passerelles utilisées au sein de la classe créée.

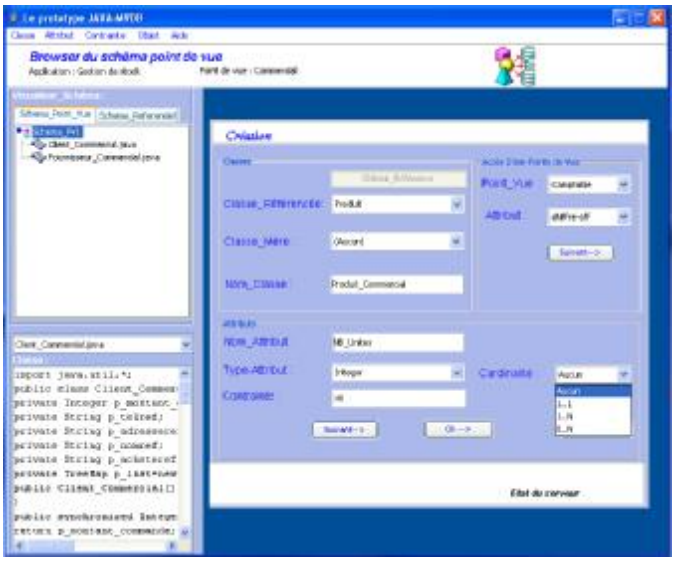


Figure 6. Le browser schéma point de vue

L'utilisation des browsers présente de nombreux avantages dont :

- Ils permettent de dégager l'utilisateur de tout apprentissage d'un langage de définition de schémas.
- Ils offrent une flexibilité de conception traduite par l'aspect visuel qu'ils procurent. Ainsi, toutes les informations relatives à la conception courante d'un schéma sont disponibles. Par exemple, pour créer un schéma point de vue, les classes du schéma référentiel sont visualisées en mode consultation avec toutes les informations qui leurs sont inhérentes, ce qui facilite le processus d'import des classes à enrichir dans le point de vue. Aussi, la visualisation de chaque classe d'un schéma permet de différencier les attributs propres de la classe de ceux importés du schéma référentiel ou de schémas points de vue par une couleur distincte.
- Les browsers offrent une vérification automatique de quelques erreurs de conception. Les contrôles qui ont été implantés sont les suivants :
  - ü La gestion des conflits structurels des attributs et des classes: homonymie et synonymie.
  - ü La suppression d'une classe dans le schéma référentiel provoque un accès automatique aux sites des différents points de vue pour une demande de suppression de la classe d'extension correspondante.
  - ü La suppression de tout attribut d'une classe d'un schéma point de vue faisant partie d'un accès inter-points de vue provoque un accès automatique aux sites des différents points de vue pour une demande de suppression de l'attribut dans la classe d'extension correspondante.

Les browsers disposent d'une capacité de stockage des schémas dans un format XML qui permet l'import et l'export des schémas entre les différents sites du système distribué d'une part, et d'une autre part il facilite la manipulation et l'accès direct et précis aux éléments composants un schéma en utilisant les primitives d'accès de XPATH. Ces accès sont nécessaires dans la conception automatique des schémas.

2.3.3. Génération des schémas en JAVA

Dans cette section, nous décrivons le processus de génération automatique des schémas dans MVDB.

2.3.3.1. Principe

Générer un schéma MVDB consiste à générer automatiquement plusieurs fichiers (Voir Figure 7). En effet, pour toute entité du domaine, nous allons créer une classe JAVA persistante de même nom. La persistance engendre (après la compilation du fichier JAVA) une création automatique du fichier d'extension .ser et de même nom où sera stockée sur disque l'extension de la classe. Nous utilisons le mécanisme de paquetage de JAVA pour regrouper toutes les classes d'un schéma dans un même espace disque.

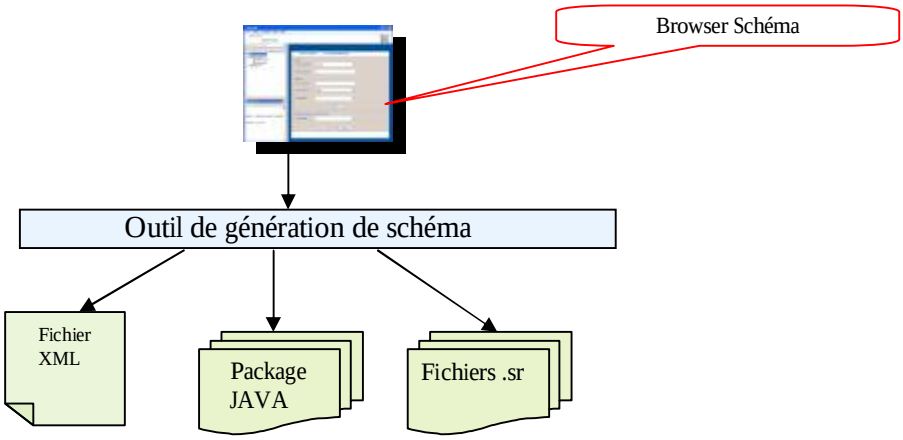


Figure 7. Outil de génération d'un schéma à partir d'une conception en MVDB

Parallèlement au processus de création du code en JAVA, un fichier XML du nom du schéma est construit automatiquement. Ce fichier contient la description de tout le schéma objet sous le format XML. Le prototype dispose d'un outil qui permet, à l'instar du mécanisme d'introspection de JAVA, d'offrir une panoplie de méthodes de recherche d'éléments. Cet outil utilise les primitives du parseur JDOM et DOM4J. Les fichiers XML sont principalement utilisés pour :

- Faciliter le parcours d'arbres des différents schémas.
- Echanger les informations entre les différents sites.

2.3.3.2. Génération des classes

Générer un schéma objet consiste à générer une hiérarchie de classes. Dans MVDB, nous distinguons le schéma référentiel et les schémas points de vue. Dans un schéma référentiel, pour toute entité du domaine, nous créons automatiquement une classe en JAVA. Toutes les classes mères implantent des interfaces sérialisables afin d'assurer la persistance des objets. La figure 8 présente le fichier client.java contenant le code en JAVA crée pour la classe "client", sous-classes de "personne".

```
Package schema.Referenciel;
import java.util.*;
public class Client extends Personne{
private String p_achete;
private TreeMap p_list=new TreeMap();

public Client(){}

public synchronized String getP_achete(){
   return p_achete;
}

public synchronized void setP_achete(String p_achete){
   this.p_achete=p_achete;
}
public synchronized void setP_list(Integer cle,String adresse){
     if ((p_list.containsValue(cle+adresse))==false){
          p_list.put(cle,adresse);
     }
}
public synchronized TreeMap getP_list(){
   return p_list;
}
}
```

Figure 8. Le fichier client.java dans le package "schema Referentiel"

L

Le fichier XML du schéma référentiel de l'application est présenté en figure 9.

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema_referenciel>
<Personne p_nom="String" p_adresse="String" p_tel="String" cle="p_nom"  >
<Fournisseur p_fournit="List" typeList="Produit"/>
<Client p_achete="List" typeList="Produit"/>
</Personne>
<Produit p_designation="String" p_prix="Integer" p_clients="Integer"
p_fournisseurs="List"
typeList="Fournisseur" cle="p_designation"  >
 p_fournisseurs="Integer">
<P_GrandPublic p_ref="Integer" />
<P_Professionnel p_ref="Integer"/>
</Produit>
<Bon p_num="Integer" cle="p_num">
<B_Commande p_BL="List" typeList="B_Livraison"/>
<B_Livraison p_progression="String" p_BC="List" typeList="B_Commande"/>
</Bon>
</Schema_referenciel>
```

Figure 9. Le fichier XML du schéma référentiel de l'application "GPDP"

La conception d'un schéma point de vue suit le même principe en respectant les règles de structuration présentées au chapitre 3. En effet, les entités d'un schéma point de vue peuvent être liées par les relations vp-extension ou vp-dependency aux classes du référentiel ou aux classe des points de vue, respectivement. Cependant, afin d'implanter le mécanisme d'héritage entre objets ou encore la délégation présenté au §1.2.4 du chapitre précédent, nous reprenons dans toute classe du schéma point de vue, la description des attributs issus d'une classe de base ou d'une passerelle. Cette reprise s'effectue automatiquement à partir du fichier XML transmis à chaque site à la création d'un schéma point de vue. La figure 10 présente le fichier JAVA de la classe "client_commercial", extension de la classe "client" présentée ci-dessus selon le point de vue commercial. Les attributs p_telref, p_adresseref; p_nomref et p_acheteref sont hérités de la classe "client". p_montant_commande est un attribut propre. A la création d'un objet, seuls les attributs propres seront instanciés, p_montant_commande dans l'exemple. Les valeurs des attributs hérités seront obtenues par invocation à distance des méthodes appropriées. Les valeurs des attributs p_telref, p_adresseref; p_nomref et p_acheteref sont obtenues par invocations des méthodes getP_tel(), getP_adresse(), getP_nom() et getP_achete() définies dans la classe "client" du référentiel.

```
Package schema.PV.Commercial;
import java.util.*;
public class Client_Commercial implements java.io.Serializable{
private Integer p_montant_commande;
private String p_telref;
private String p_adresseref;
private String p_nomref;
private ArrayList p_acheteref;
private TreeMap p_list=new TreeMap();
public Client_Commercial(){}
public synchronized Integer getP_montant_commande(){
   return p_montant_commande;
}
public synchronized String getP_telref(){
   return p_telref;
}
public synchronized String getP_adresseref(){
   return p_adresseref;
}
public synchronized String getP_nomref(){
   return p_nomref;
}

public synchronized ArrayList getP_acheteref(){
   return p_acheteref;
}
public synchronized void setP_montant_commande(Integer p_montant_commande){
   this.p_montant_commande=p_montant_commande;
}
public synchronized void setP_telref(String ref_valeur){
   this.p_telref=ref_valeur;
}
public synchronized void setP_adresseref(String ref_valeur){
   this.p_adresseref=ref_valeur;
}
public synchronized void setP_nomref(String ref_valeur){
   this.p_nomref=ref_valeur;
}
public synchronized void setP_acheteref(ArrayList ref_valeur){
for(Object obj:ref_valeur){
   this.p_acheteref.add(obj);
}
}
public synchronized void setP_list(Integer cle,String adresse){
   p_list.put(cle,adresse);
}
public synchronized TreeMap getP_list(){
   return p_list;
}
}
```

Figure 10. Le fichier Client_Commercial.java dans le package " schema.PV.Commercial "

2.3.3.3. Génération des attributs

Dans MVDB, les accès à tous les attributs sont décrits aux moyens de fonctions accesseurs. En effet, pour un attribut de type simple (littéral) de nom "nomatt" deux accesseurs sont automatiquement générés: **getNomatt** pour l'accès en lecture et **setNomatt** pour l'accès en écriture. Par contre, pour un attribut de type référencé (association) de nom "nomatt", trois accesseurs sont générés: **getNomatt** pour l'accès en lecture, **addNomatt** et **removeNomlatt** pour l'accès en écriture. L'accès aux valeurs des attributs uniquement via des fonctions accesseurs présente plusieurs avantages dont:

- L'encapsulation des données favorise une meilleure programmation en réduisant les erreurs d'une mauvaise utilisation des données.
- Le nommage des accesseurs respecte le protocole des javabeans, c'est-à-dire les accesseurs en lecture sont préfixés par "get" et ceux en écriture par "set", ce qui permet d'unifier les accès et faciliter ainsi la génération automatique des accesseurs.
- La manipulation des attributs référencés est conforme à la spécification ODMG 2.00 [**]. Elle ne peut se faire que par les accesseurs **get**, **add** et **remove**. Ce qui permet la vérification des contraintes de cardinalité et garantit le mécanisme d'intégrité référentielle.

Pour assurer une gestion des accès concurrents aux attributs, tous les accesseurs sont déclarés **synchronized**. Aussi, ils sont **public** et peuvent donc être invoqués par les clients de chaque schéma. Cependant, notons que les accesseurs de tout attribut sont implantés dans la classe de son introduction et sont de type **static**. Tout accès externe (via un accesseur au même attribut spécifié dans une classe d'un autre schéma) est réalisé via un service à distance (CORBA). La figure 11 montre un accès via une requête.

```
public String get_Objects(String requete){
String reponse=null;
Serveur    serveur_ref=new
Client_Point_Vue().getServeurcomm(machine,port,nomService);//service
       try{
       reponse=serveur_ref.get_Objects(requete);//String

      ser.XML_serializer("C:/Schema_Commercial/XML/"+requete+"s.xml",reponse);

       }catch(Exception e){
       System.err.println("Erreur"+e);

       }
       return reponse;

  }
```

Figure 11. Exemple d'invocation de méthode à distance

2.3.3.4. Opérations sur les schémas

Outre la création des classes des différentes hiérarchies du référentiel et des points de vue, les browsers disposent de certaines opérations de mise à jour. Cependant, notons que la modification d'un schéma multi-points de vue considère des techniques d'évolution des schémas qui relève d'un autre aspect des bases de données qui sort du cadre de notre travail. Nous ne considérons ici que quelques opérations simples de modification d'un schéma multi-points de vue (schéma référentiel et schémas points de vue). Aussi, ces modifications ne concernent que les schémas dont les bases d'objets sont vides.

Les opérations sur les schémas disponibles dans la version actuelle du prototype sont : créer une classe, supprimer une classe, renommer une classe, modifier la classe mère d'une classe, modifier la classe de référence dans une classe d'un schéma point de vue, supprimer un attribut, modifier le type d'un attribut, renommer un attribut, modifier la cardinalité d'un attribut.

2.4. Browsers d'objets

Les différents paquetages générés par les browsers des schémas présentés ci-dessus peuvent être utilisés dans un programme applicatif JAVA pour créer des objets et gérer des bases de données multi-points de vue. Cette utilisation se traduit par leur importation dans le programme applicatif (clause Import). Un exemple d'utilisation du package "schema.PV.Commercial" est présenté en figure 12. Cependant, le prototype JAVA-MVDB offre aux utilisateurs deux browsers de création des objets de base et des objets points de vue. L'utilisation des browsers présente les mêmes avantages cités en §2.3.2.

```
import schema.PV.Commercial.* ;
import java.util.* ;
public class Client_web{
Client_Commercial client=new Client_Commercial() ;
 public Client_web(){}
 public Ajouter_client(String nom,String adr,String tel,Integer montant,ArrayList produit)
{
try
{
 client.setP_nomref(nom);
client.setP_adresseref(adr) ;
client.setP_tel(tel) ;
client.setP_montant_commande(montant) ;
client. setP_acheteref(produit);
Enregistrer_client(client);
}catch(Exception_Client e){
Exception_Client.excep_type(e);
}
}
Public Enregistrer_client(Client_Commercial client)
{
   ……... ……
    ……
}
……..
```

Figure 12. Exemple d'exploitation d'un package

2.4.1. Browser des objets de base

Ce browser, présenté en figure 13, contient toutes les données relatives aux objets multi-points de vue. Il est constitué d'une zone principale et d'une zone d'édition. La zone d'édition est située à gauche et se compose d'une partie supérieure qui affiche la hiérarchie des classes du référentiel. La sélection d'une classe permet l'affichage dans la partie inférieure de l'ensemble ses objets. La zone principale permet l'affichage des différents écrans relatifs aux différentes opérations disponibles pour les objets (création, suppression, modification ..). La consultation d'un objet permet d'afficher sa classe d'instanciation, les valeurs de tous ses attributs ainsi que la liste des objets points de vue auquel il est lié.

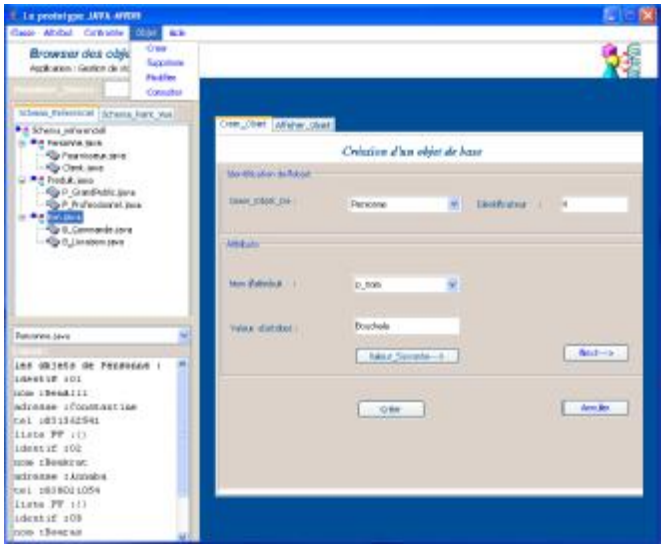


Figure 13. Le browser des objets de base

2.4.2. Browser des objets points de vue

Ce browser contient toutes les données relatives aux objets d'une base point de vue. Son interface, présentée en figure 14, ne diffère pas de celle du browser des objets de base en ce qui concerne la zone d'édition sauf que les objets affichés sont ceux des classes du schéma point de vue. Cependant, la création d'un objet point de vue nécessite d'indiquer l'objet de base auquel il est associé. Le choix de cet objet n'est pas arbitraire comme présenté en §2.2.3. du chapitre 3. En appliquant un algorithme de sélection des objets candidats, le browser affiche une liste qu'il met à la disposition de l'utilisateur pour sélectionner l'objet de base à étendre dans ce point de vue. La consultation d'un objet point de vue permet d'afficher sa classe d'instanciation, les valeurs de tous ses attributs hérités et propres ainsi que l'objet de base qu'il étend.

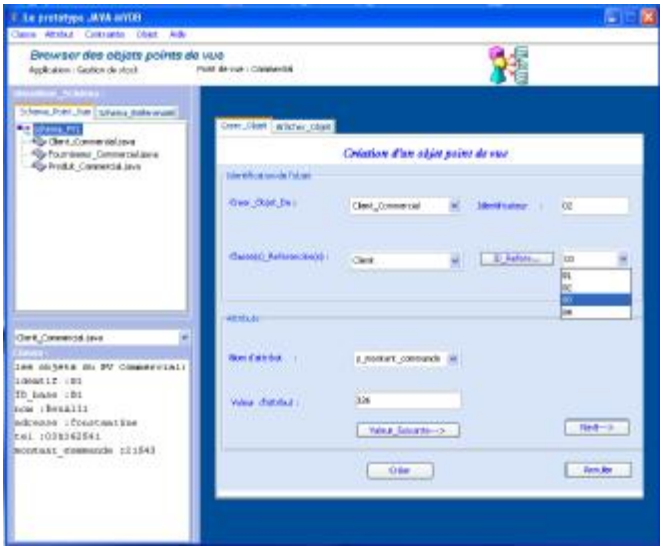


Figure 14. Le browser des objets points de vue

2.5. Bilan et validation de l'approche

Pour valider notre approche, nous avons choisi une application simple de gestion et de distribution des produits dans une entreprise de fabrication. Ce choix est justifié par le fait que le but principal est de présenter sur un exemple simple les concepts de notre modèle d'une part et de mettre en œuvre un prototype de conception distribuée des bases de données multi-points de vue d'une autre part en favorisant:

- Une représentation multiple des entités du domaine selon des points de vue différents.
- Une indépendance de conception au niveau des différents sites moyennant des règles inhérentes à la modélisation des objets dans MVDB et ceci afin d'assurer la cohérence globale des objets.
- L'interopérabilité et la coopération des concepteurs en échangeant leurs données.
- La protection et la cohérence des données par la mise en œuvre de mécanismes appropriés tel que le mécanisme de délégation et celui des contraintes d'intégrité.

D'une manière générale, notre prototype peut s'appliquer sur des projets plus complexes dans lesquels une unique description des objets est insuffisante pour manipuler et exploiter ces derniers. Les applications de Conception Assistée par Ordinateur (CAO) et les Systèmes d'Informations Géographiques (SIG) en sont des exemples type. Dans ces derniers, la réalisation d'un projet exige l'intervention de plusieurs spécialistes qui se partagent la tâche de conception. Nous considérons que ces applications sont un terrain privilégié pour nos futures expérimentations.

Dans la version actuelle du prototype, bien que le noyau du modèle a été réalisé, tous les concepts de base et les mécanismes associés sont implantés, différentes parties sont en cours de développement notamment en ce qui concerne :

- Le contrôle des contraintes d'intégrité globales.
- Les opérations de mise à jour des schémas avec des contrôles de granularité très fine.
- Le contrôle des accès inter-points de vue.

Enfin, il convient de souligner que l'implémentation réalisée constitue plus une étude de faisabilité qu'une application aboutie. Elle a cependant été fort utile comme support à la réflexion en permettant de mettre en évidence les concepts et les aspects essentiels de notre approche.

3. Extension de l'approche : une architecture support à MVDB

L'approche prônée en MVDB se propose de prendre en compte le mécanisme des points de vue de la conception à l'exploitation d'un système d'information. Une fois un schéma multi-points de vue conçu d'une façon distribuée selon des points de vue différents, quelle architecture distribuée doit-on adopter afin d'assurer sa gestion tout en préservant la spécificité des points de vue?

Différentes architectures sont proposées pour l'intégration des bases de données dans un même système distribué allant des systèmes fortement couplés vers les systèmes faiblement couplés ou les systèmes fédérés. Dans cette section nous présentons deux extensions de l'approche multi-points de vue. Dans un premier temps, nous proposons une architecture support à l'approche après une présentation succincte des architectures existantes. L'intégration de nouveaux composants est une caractéristique requise pour tout système afin d'assurer son extensibilité. Nous proposons ensuite une démarche d'intégration d'un nouveau composant déjà existant en exploitant les concepts mêmes du modèle MVDB.

3.1. Intégration des bases de données

L'intégration des données est un sujet de recherche très actif depuis plusieurs années. L'objectif initial, était de pouvoir gérer des données manipulées par des propriétaires et des applications différentes au moyen d'un système centralisé de gestion de bases de données. Des besoins incessants se sont faits sentir par la suite pour regrouper l'ensemble des données qui sont gérées par des SGBDs différents. Les premiers travaux sur les systèmes multibases des données ont été proposés par [48]. Le terme fédération de bases de données a été introduit par la suite afin de décrire des systèmes qui ont pour tâche de fournir un moyen pour pouvoir accéder d'une manière unifiée aux données qui sont stockées dans diverses sources. Depuis la recherche sur les bases de données fédérées est devenue un domaine très actif caractérisé par des techniques permettant un accès intégré à des bases de données distribuées, hétérogènes et autonomes ainsi qu'en témoignent les travaux de Sherth et Larson [81]. Ces auteurs ont effectué une synthèse des résultats qui sont obtenus dans ce domaine et ont proposé une référence de classification la plus communément retenue pour les systèmes distribués. Ces derniers sont divisés en deux familles: les SGBDDs non fédérés ou fortement couplés et les SGBDDs fédérés ou faiblement couplés.

3.1.1. Les SGBDs fortement couplés

Les systèmes fortement couplés ont un schéma global unifié accédé par les utilisateurs. Ainsi, tous les schémas des différentes ressources sont intégrés dans un seul schéma global. Différentes méthodologies pour l'intégration des schémas sont proposées. Nous renvoyons le lecteur aux travaux de Batini et al. [66] pour une étude détaillée et comparative de ces méthodologies.

Le processus d'intégration devrait alors supprimer toutes les incohérences, les erreurs et les redondances issues des différents schémas. Cependant, cette intégration des schémas fait perdre à ces derniers toute leur autonomie. En effet, il n'existe qu'un seul niveau de gestion où toutes les opérations sont exécutées d'une façon uniforme. Aucune distinction n'est alors faite entre utilisation locale et utilisation globale des données.

Nous remarquons que l'objectif de l'intégration dans ces systèmes n'est pas de prendre en compte les descriptions des utilisateurs pour construire un schéma conceptuel structuré relativement à ces derniers. Le schéma conceptuel résultant de l'intégration ne diffère pas dans sa forme de celui issu d'une conception directe. Les spécificités relatives aux points de vue des utilisateurs sont perdues à l'issue de la conception par intégration. Cette approche ne répond donc pas aux besoins de structuration d'un schéma de bases de données selon des points de vue différents exprimés en §****. Nous verrons dans ce qui suit que l'approche des systèmes fédérés, par contre, permet de préserver cette description multi-points de vue des données.

3.1.2. Les SGBDs faiblement couplés

Les systèmes faiblement couplés n'ont pas un schéma global mais offrent en général un langage commun pour l'accès aux données. Dans les travaux sur l'intégration des BDDs dans un système faiblement couplé, la nouvelle base de données ne contient pas d'informations propres. Elle exploite les informations issues des composants à la façon des schémas vues. Cette base de données est désignée sous le nom de base de donnée fédérée. Un système de bases de données fédérées offre à ses clients la possibilité de percevoir et de manipuler de façon transparente les données de l'ensemble des composants décrites par un schéma intégré, appelé schéma fédéré. Scheth [81] indique q'une fédération peut également détenir ses propres informations qui sont des méta-données utilisées pour assurer une bonne gestion des composants. Parmi les objectifs principaux de la fédération, nous citons :

- Assurer la transparence vis-à-vis des clients de la base de données fédérée.
- Permettre la récupération, lors de l'exploitation, des données issues des différents composants.
- Une fois intégrées, les ressources doivent continuer à mener leur propre existence.
- Préserver l'autonomie des composants, c'est-à-dire que l'intégration doit être transparente pour leurs clients.

Nous constatons que ces objectifs suivent directement la perspective et la motivation principale de l'approche multi-points de vues à laquelle nous nous intéressons. Nous proposons donc l'adoption d'un tel environnement pour le support de l'intégration de bases de données décrivant un même univers de discours selon des points de vue différents.

Les extensions que nous proposons pour notre approche s'inspirent essentiellement de l'architecture de référence des systèmes fédérés proposée dans [81] et de quelques techniques d'intégration utilisées dans le cadre du modèle "objet". Nous commençons d'abord par donner un bref aperçu de l'existant avant de détailler nos propos.

3.1.3. Architecture d'une base de donnés fédérée

L'architecture adoptée par la plupart des travaux dans le domaine des bases fédérées est donnée dans [81]. A l'image de l'architecture ANSI/X3/SPARC [ANSI75] à trois niveaux, l'architecture de [81] contient cinq niveaux. La figure 15 illustre cette architecture à cinq niveaux.

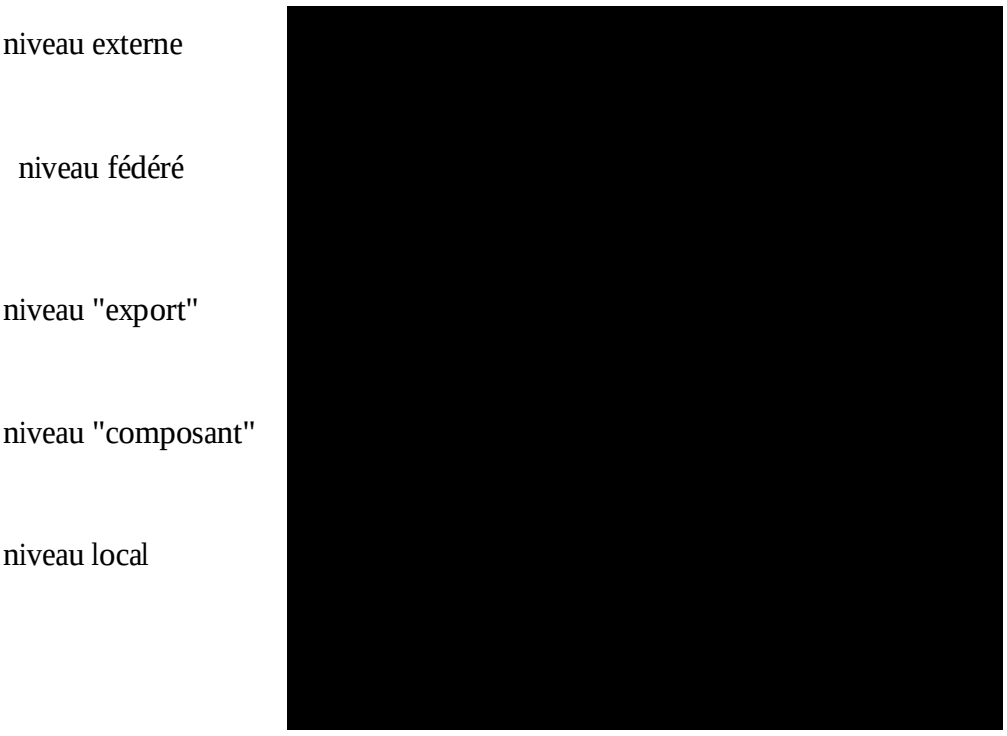


Figure 15. Architecture à cinq niveaux d'une base de données fédérée

- *niveau local* : niveau composé des schémas des composants (encore appelés *schémas locaux) ;*

- *niveau "composant"* : niveau composé des schémas "composants". Un schéma "composant" est un schéma local traduit dans son intégralité dans le modèle de données commun. Entre le niveau local et le niveau "composant", un processeur de transformation assure la traduction des requêtes et des données dans les deux sens;

- *niveau "export"* : niveau composé des schémas exportés. Un schéma exporté est un schémavue d'un composant. Il est exprimé dans le modèle de données commun. Il décrit l'ensemble des informations d'un composant pertinentes pour la fédération. Un processeur de filtrage assure l'extraction des informations pertinentes entre le niveau "composant" et le niveau "export" ;

- *niveau fédéré* : le schéma fédéré est le résultat de l'intégration complète des schémas exportés. On peut donc considérer qu'il s'agit aussi d'un schéma-vue sur l'union des

schémas exportés. Un processeur de construction s'occupe de la distribution des requêtes provenant des clients de la base fédérée et de l'intégration des résultats issus des ces requêtes distribuées. Lors de l'exploitation, il assure l'articulation entre le niveau fédéré et le niveau "export" ;

- *niveau externe* : niveau composé de schémas-vues externes. Un schéma-vue du niveau externe décrit les informations de la fédération pertinentes pour un de ses clients. On retrouve un processeur de filtrage entre le niveau fédéré et le niveau externe.

Tous les niveaux de cette architecture ne sont pas forcément tous présents dans une base de données fédérée.

3.2. Architecture support au modèle MVDB

L'architecture support au modèle MVDB consiste en une collection de bases de données locales partielles qui coopèrent dans un environnement fédéré pour la gestion des entités d'un même univers de discours. En effet, suivant l'approche point de vue, le schéma global est conçu d'une façon décentralisée selon plusieurs points de vue. Chaque point de vue est détenu par une base de données autonome. Cette autonomie favorise l'indépendance des bases composantes. Elle permet à chacune d'elles de garder une description complète des entités selon un point de vue donné. Ainsi, les entités sont décrites de façon multiple mais complémentaire par plusieurs schémas qui partagent une description de base. Dans le paragraphe suivant, nous présentons une description générale de cette architecture. L'objectif est d'offrir une plate forme d'expérimentation pour une description multiple et une exploitation des entités d'un même univers de discours.

3.2.1. Description générale

L'architecture que nous proposons, présentée en figure 16, se compose de trois niveaux : le niveau *local*, le niveau *fédéré* et le niveau *externe*. Le niveau *local* supporte les différentes bases points de vue. Le niveau *fédéré* permet la gestion globale de la base multi-points de vue dont les services sont exploités par le niveau *externe*. Le modèle commun utilisé est le modèle objet. Les problèmes de l'hétérogenité ne sont pas traités. Le niveau "composant" et le niveau "export" de l'architecture de référence sont omis. En effet, tous les schémas locaux sont déjà exprimés dans le modèle commun (objet) et se situent donc au niveau "composant". Le niveau "export" n'est pas nécessaire dans notre approche car les composants exportent l'intégralité de leurs données.

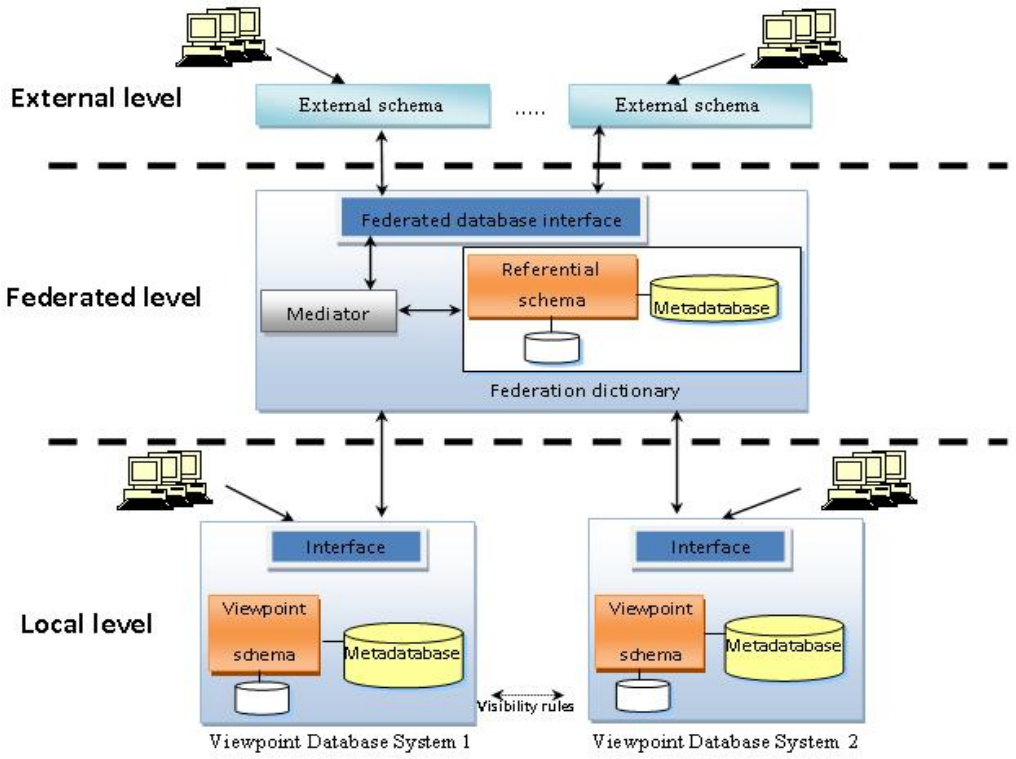


Figure 16. Architecture support au modèle MVDB

3.2.2. Description des différents composants

Nous décrivons d'une manière générale les différents composants de l'architecture proposée.

Niveau fédéré : c'est le noyau qui assure la gestion du système fédéré. Il est essentiellement composé de :

- **Interface de la base de données fédérée** (Federated_database_Interface)) : elle permet la communication avec les utilisateurs du niveau externe. Elle leur assure une transparence totale qu'en à la participation des bases points de vue au niveau local.
- **Dictionnaire de la fédération** (Federation_dictionary) : c'est un module qui contient le référentiel et une metadatabase. Le référentiel est le schéma de base étendu par les différentes bases de données au niveau local. La metadatabase est un composant dont l'importance est indéniable. Il contient deux types d'informations : celles relatives aux types de données supportées par les différentes bases de données oints de vue au niveau local et les contraintes globales. Ces dernières permettent le maintien de la cohérence globale de la base fédérée et la résolution des conflits d'intégration à l'exploitation. La metadatabase est utilisée par le médiateur lors du traitement des requêtes des utilisateurs.

- **Médiateur (Mediator) :** c'est un processeur qui permet l'évaluation des requêtes des utilisateurs de la base de données fédérée. Ainsi, lors de la réception d'une requête, il vérifie sa validité en se basant sur les informations stockées dans la metadatabase, par exemple, toute requête qui accède à deux bases de données exclusives est rejetée. Une fois validée, un plan d'exécution est établi et la requête est divisée en sous)requêtes. Le médiateur s'occupe ensuite de la distribution des sous-requêtes et de l'intégration des résultats issus des bases de données locales

Niveau local : dans ce niveau, les BDDs partielles sont interdépendantes. Cette interdépendance est régit par des règles de visibilité qui expriment les accès inter-points de vue gérés par des métadatabases. Ces dernières permettent d'assurer la communication entre les BDDs.

Niveau externe : ce niveau concerne l'exploitation de la base de données multi-points de vue fédérée. Un mécanisme de vue peut être définis pour le modèle MVDB et qui permet de spécifier des schémas externes à l'instar des vues dans une base de données mono-points de vue.

4. Conclusion

Dans ce chapitre, nous avons présenté une mise en œuvre du modèle MVDB réalisée en JAVA. Le principe de génération automatique des schémas (classes et attributs de classes) a été détaillé ainsi que le mécanisme de partage de propriétés entre les objets. La persistance des objets est réalisée via le mécanisme de sérialisation offert par le langage JAVA. Cependant, il serait souhaitable de réaliser notre prototype en se basant sur un SGBDDO existant tels que "Objectstore", "Gemstone" ou encore "Poet" et l'étendre par une couche supportant le mécanisme des points de vue. Ainsi, nous aurions pu user de toutes les fonctionnalités de base des SGBDDO telles que la persistance, la gestion des extensions, les requêtes, la sécurité, etc. Cependant la non disponibilité de ces SGBDDO nous a conduit à construire notre système "from scratch".

L'architecture proposée est en cours d'étude et de développement. Sa mise en œuvre va permettre de gérer une base de données distribuée et fédérée dont les objets ne sont pas complètement indépendants. Cependant, une gestion autonome des données est réalisée au niveau de chaque site moyennant quelques contraintes inhérentes à la nature du système.

Conclusion générale et perspectives

1. Rappel de l'objectif de la thèse

L'approche par points de vue à laquelle nous nous intéressons a ouvert de nouveaux horizons sur la manière de modéliser le monde réel avec toute la complexité, l'irrégularité et la richesse qu'il représente. Les objectifs de la thèse se résument à définir un modèle de données compatible avec le standard de l'ODMG intégrant le concept de point de vue et à implémenter un outil pour la conception des bases de données multi-points de vue cohérentes et distribuées. En effet, le modèle de données multi-points de vue, MVDB, proposé permet à un schéma d'une base de données d'être une représentation multiple d'un univers de discours. Il permet une co-existence de plusieurs représentations, il respecte la spécificité de chacune et favorise l'échange, le partage et la protection des informations entre elles. De plus, il fournit un langage pour spécifier des contraintes d'intégrité garantissant la cohérence de la représentation multiple. Dans MVDB, la notion de point de vue est inhérente au modèle. Un point de vue est représenté par un schéma et une base d'objets

2. Bilan du travail réalisé

Notre travail a été introduit en présentant une problématique concernant les nouveaux besoins en structuration des données qui s'imposent avec l'ouverture des bases de données vers les nouvelles applications de complexité accrue d'une part et l'expansion des systèmes distribués d'une autre part. Nous avons mis en évidence la nécessité de doter un modèle de bases de données d'une capacité de représentation ayant quatre caractéristiques principales : (1) multiple, (2) distribuée (décentralisée), (3) possédant un référentiel (une représentation de base) et des représentations partielles associées, (4) permettant un échange d'informations entre les représentations partielles, et enfin (4) cohérente.

ü Nous avons mené, dans le chapitre 1 et 2, une étude des travaux qui se sont intéressés à la représentation multiple dans des contextes différents et plus particulièrement dans le domaine des bases de données. En particulier, les concepts de vues et de rôles qui tentent de modéliser les points de vue sont abordés et une étude comparative des trois mécanismes de vue, rôle et point de vue est faite à la fin du chapitre 2 afin de positionner l'approche prônée dans MVDB.

ü Ensuite, nous avons défini dans le chapitre 3, le modèle multi-points de vue MVDB dans lequel tout concept de base du modèle objets (schéma, objet) est une représentation multiple. De plus, la représentation multiple est décentralisée, possède un référentiel, favorise l'échange d'information entre les représentations partielles via le concept de "passerelle" et elle est cohérente. Ces deux derniers aspects ont été présentés dans le chapitre 4. Deux types de contraintes d'intégrité sont distinguées dans le cadre de notre modèle : les contraintes intra-schéma, et les contraintes inter-schémas. En se basant sur les concepts de base de la logique du premier ordre, nous avons proposé un langage d'expression de ces contraintes.

ü Enfin, nous avons réalisé une mise en œuvre du modèle. Les différentes fonctionnalités du système java-MVDB sont présentées au chapitre 5. java-MVDB intègre et valide les notions (pas toutes?) développées tout au long de ce document.

Nous avons conclu notre travail par la proposition d'une architecture de bases de données fédérées pour la gestion des objets multi-points de vue.

3. Perspectives

Au terme de cette thèse, nous pouvons envisagé en perspectives un certain nombre d'extension de note travail.

Extensions du modèle

Plusieurs extensions du modèle peuvent être enregistrées :

- Tout d'abord, étant donné que notre modèle est défini pour les bases de données, il faudrait compléter notre proposition par un langage de requêtes, extension du langage OQL, qui tienne compte du fait que les objets sont des objets multi-points de vue et qu'ils peuvent donc être interrogés suivant un ou plusieurs points de vue.
- Par ailleurs, il serait intéressant de compléter notre étude sur les contraintes d'intégrité par une méthode de vérification qui permet le maintien de la cohérence d'une base multi-points de vue.
- Dans cette thèse, l'aspect comportemental des objets n'est pas pris en compte. En effet, nous rappelons que nous avons considéré des objets serveurs d'information et n'avons pas développé l'aspect "traitement de l'information". Un atout de l'approche "objet" par rapport à l'approche "relationnelle" est l'intégration du langage de programmation au sein du SGBD, ce qui donne lieu à un regroupement des aspects "données" et des aspects comportementaux au sein des objets de la base de données. Dans le contexte de notre travail, nous pouvons, par exemple, exploité l'usage des méthodes pour prendre en compte les besoins d'interaction entre les points de vue.

Extension du prototype

En ce qui concerne le prototype, nous devons signaler que certaines des fonctionnalités présentées dans ce document ne sont pas encore intégrées complètement, ils sont en cours de développement, notamment :

- La spécification des contraintes d'intégrité globales.
- Les opérations sur les schémas (suppression et modification dans la structure des classes).

Cependant, nous envisageons d'étendre ce prototype par un module d'exploitation des données qui permet d'offrir des opérations élémentaires de recherche dans une base de données multi-points de vue (trouver les différentes représentations partielles d'un objets, accéder aux données d'un objet selon un point de vue, ..).

Nouveaux horizons

Nous avons proposé une architecture basée sur la fédération pour la gestion des bases de données multi-points de vue. Cet aspect ouvre de nouveaux horizons vers des applications très intéressantes intégrant le concept de point de vue dans les bases de données fédérées, notamment :

- Etudier la possibilité d'intégration d'un nouveau composant "base de données" lors de la conception d'une base de données multi-points de vue en MVDB en

prenant comme hypothèse que ce composant correspond à un point de vue particulier de l'univers modélisé. Pour ce faire, des méthodes d'intégration par vues [2] [77] [78] [79] [80] peuvent être adaptées et exploitées.

- Généraliser notre approche multi-points de vue pour la conception de bases de données fédérées multi-points de vue en partant exclusivement d'un ensemble de composants existants sans imposer que ceux-ci correspondent aux points de vue sur un univers de discours. Le but d'une telle approche serait alors de "retrouver" les schémas points de vue au sein de cet ensemble de composants par rapport à un référentiel commun. Ce référentiel pourrait être déterminé à partir des descriptions présentes dans les schémas des différents composants.

Ces quelques commentaires montrent que le sujet est encore loin d'être épuisé. Nous envisageons donc de continuer à explorer les différentes possibilités encore ouvertes.

Annexe A

Etude de cas : Gestion d'une Entreprise de Fabrication et de Distribution des Produits (GEF-DP)

1. Présentation

Pour valider notre approche, nous avons choisi une application simple de gestion d'une entreprise industrielle qui achète, fabrique et distribue ses produits. Les entités du domaine sont : les clients, les fournisseurs, les produits, les entrepôts et les bons (de commande et de livraison). Nous supposons que la réalisation de cette gestion requiert le concours de plusieurs intervenants qui se partagent la tâche de conception et interagissent simultanément sur la manière de représenter les entités du domaine. Les différents intervenants représentent les points de vue et se basent sur une description de base des entités du domaine « le référentiel » et l'étendent chacun selon la fonction qu'il exerce dans l'entreprise. Nous considérons quatre points de vue : commercial, de fabrication, de direction et comptable.

Dans ce qui suit, nous présentons la description en ODL adapté pour nos besoins des différents schémas de la base de données : schéma référentiel et ses extensions selon les différents points de vue.

2. Description du référentiel

Les entités du domaine sont :

- les partenaires externes décrits par la classe "part-externe",
- les clients décrits par la classe "client", sous classe de "part-externe",
- les fournisseurs décrits par la classe "fournisseurs", sous classe de "part-externe",
- les produits décrits par la classe "produit",
- les produits grand public décrits par la classe "produit-grand-public",
- les produits professionnels décrits par la classe "produit-professionnel",
- les bons décrits par classe "bon",
- les bons commande décrits par la classe "bon-commande",
- les bons de livraison décrits par la classe "bon-livraison", et les entrepôts décrits par la classe "entrepôt".

Le schéma référentiel exprimé en ODL est le suivant:

```
class  Part-externe (extent part-externes key nom, telephone)
{
    attribute string nom;
    attribute string telephone;
    attribute string adresse;
```

```
}

class  Client : Part-externe(extent clients key nom, telephone) {
      relationship set<Produit> a_achete inverse Produit::a_été_acheté_par;
}

class  fournisseur : Part-externe(extent fournisseurs key nom, telephone) {
      relationship set<Produit> fournit inverse Produit::est_fourni-par;
}

class  Produit (extent produits) {
      attribute string designation;
      attribute integer prix_unitaire;
      relationship set<Client> a_été_acheté_par inverse Client:: a_achete est;
      relationship set<Fournisseur> est_fourni-par inverse Fournisseur:: fournit;
}

class  Produit_professionnel : Produit (extent produits_prof
                                        key reference_professionnelle) {
      attribute string reference_professionnelle;
}

class  Produit_grand_public :Produit (extent produits_grand_public
                                      key reference_grand_public) {
      attribute string reference_grand_public;
}

class  Entrepot (extent entrepots  Key nom)
     {
      attribute string nom;
      attribute string adresse;
      attribute integer capacite;
     }

class  Bon_de_livraison (extent bons_livraison key numero)
     {
     attribute integer numero;
     attribute string progression;
     relationship set<Bon_de_commande>[1..1] correspond inverse Bon_de_commande::correspond; }

class  Bon_de_commande (extent bons_commande key numero)
     {
     attribute string numero;
     relationship set<Bon_de_livraison>[1..1] correspond inverse Bon_de_livraison::correspond;
     }
```

3. Description du point de vue "comptable"

```
class  Part-externe_comptable extend Part-externe (extent part-externes_comptable key nom, telephone)
{
     referentiel part :
          attribute string nom;
          attribute string telephone;
          attribute string adresse;
}

class  client_comptable:Part-externe_comptable  extend  Client  (extent  clients_comptable  key  nom,
telephone)
{
  viewpoint part :
     attribute integer  en_cours_a_regler;
     attribute integer chiffre_affaire;
   referentiel part :
     relationship set<Produit_comptable>a_achete inverse Produit_comptable::a_été_acheté_par;
}

class  fournisseur_comptable:Part-externe_comptable extend fournisseur (extent
                                                    fournisseurs_comptable  key nom, telephone)
{
    viewpoint part :
     attribute integer  sommes_aquitées;
     attribute integer sommes_dues;
   referentiel part :
     relationship set<Produit_comptable> fournit inverse Produit_ comptable::est_fourni-par;
}
class  Produit_ comptable extend Produit (extent produits)
{
    viewpoint part :
     attribute integer  marge_moyenne;
     attribute integer  chiffre_affaires;
   referentiel part :
     attribute string designation;
     attribute integer prix_unitaire;
     relationship set<Client_comptable> a_été_acheté_par inverse Client_comptable:: a_achete est;
     relationship set<Fournisseur_comptable> est_fourni-par inverse Fournisseur_comptable:: fournit;
}

class  Produit_professionnel_comptable:Produit_comptable extend Produit_professionnel (extent
                                        produits_prof_comptable key reference_professionnelle) {
    referentiel part :
     attribute string reference_professionnelle;
}

class  Produit_rembourssable :  Produit_professionnel_comptable (extent produits_rembourssables
                                key reference_professionnelle) {
     attribute string garantie;
}

class  Produit_non_rembourssable :  Produit_professionnel_comptable
      (extent produits_non_rembourssables  key reference_professionnelle) {
     attribute string delai;
```

```
}
class  Entrepot-comptable (extent entrepots_comptable  Key nom)
    {
   viewpoint part :
     attribute integer  valeur_stock;
  referentiel part :
    attribute string nom;
     attribute string adresse;
     attribute integer capacite;
    }

class  Bon_de_livraison_comptable  (extent bons_livraison_comptable  key numero)
    {
   viewpoint part :
     attribute date  date_comptabilisation;
  referentiel part :
    attribute integer numero;
    attribute string progression;
    relationship  set<Bon_de_commande_comptable>  correspond  inverse
    Bon_de_commande_comptable ::correspond;
     }

class  Bon_de_commande_comptable  (extent bons_commande_comptable  key numero)
    {
   viewpoint part :
     attribute date  date_prise_en_compte;
  referentiel part :
    attribute string numero;
    relationship  set<Bon_de_livraison_comptable>  correspond  inverse  Bon_de_livraison_comptable
    ::correspond;
    }
```

4. Description du point de vue "commercial"

```
class   Part-externe_commercial  extend  Part-externe  (extent  part-externes_commercial  key  nom,
telephone) {
     referentiel part :
           attribute string nom;
           attribute string telephone;
           attribute string adresse;
}

class client_commercial:Part-externe_commercial extend Client (extent clients_commercial key nom,
telephone)
{
access-viewpoint_part Comptable:
     attribute integer encours_a_regler;
viewpoint part :
     attribute integer  montant_cmd;
     attribute date date_derniere_cmd;
 referentiel part :
     relationship set<Produit_commercial>a_achete inverse Produit_commercial::a_été_acheté_par;
}
```

```
class fournisseur_commercial:Part-externe_commercial extend fournisseur (extent
                                                    fournisseurs_commercial key nom, telephone)
{
    viewpoint part :
      attribute string contact_commercial;
   referentiel part :
      relationship set<Produit_commercial> fournit inverse Produit_ commercial::est_fourni-par;
}

class Produit_ commercial extend Produit (extent produits_commercial)
{
   access-viewpoint_part Comptable:
      attribute integer chiffre_affaires;
 viewpoint part :
      attribute integer nbre_unites_vendues;
      relationship set<Bon_de_livraison_commercial> est_contenu_dans inverse
         Bon_de_livraison_commercial ::contient;
     relationship set<Bon_de_commande_commercial> est_contenu_dans inverse
         Bon_de_commande_commercial :: contient;
   referentiel part :
      attribute string designation;
      attribute integer prix_unitaire;

}

class Bon_de_livraison_commercial (extent bons_livraison_commercial key numero)
     {
    viewpoint part :
      attribute date date_emission;
      relationship set<Produit_commercial> contient inverse Produit_commercial::est_contenu_dans-BL;
   referentiel part :
     attribute integer numero;
     attribute string progression;
     relationship set<Bon_de_commande_commercial> correspond inverse
     Bon_de_commande_commercial ::correspond;
      }

class Bon_de_commande_commercial (extent bons_commande_commercial key numero)
     {
    viewpoint part :
      attribute date date_reception;
      relationship set<Produit_commercial> contient inverse Produit_commercial::est_contenu_dans-
BC;
  referentiel part :
     attribute string numero;
     relationship set<Bon_de_livraison_commercial> correspond inverse Bon_de_livraison_commercial
     ::correspond;
     }
```

5. Description du point de vue "Fabrication"

```
class Part-externe_fabrication extend Part-externe (extent part-externes_fabrication key nom, telephone)
{
viewpoint part :
```

```
      attribute string  contact_technique;
referentiel part :
            attribute string nom;
            attribute string telephone;
            attribute string adresse;
}

class  fournisseur_fabrication:Part-externe_fabrication extend fournisseur (extent
                                                    fournisseurs_fabrication  key nom, telephone)
{
    viewpoint part :
      attribute boolean  fournit_seulement_lot_complet_de_pieces;
   referentiel part :
      relationship set<Produit_fabrication> fournit inverse Produit_ fabrication::est_fourni-par;
}

class  Produit_ fabrication extend Produit (extent produits_fabrication)
{
   access-viewpoint_part Commercial:
      attribute integer nbre_unites_vendues;
 viewpoint part :
      attribute integer  cout_fabrication;
   referentiel part :
      attribute string designation;
      attribute integer prix_unitaire
}

class  Produit_professionnel_fabrication : Produit_fabrication (extent produits_prof _fabrication
                                        key reference_professionnelle)
{
  viewpoint part :
      attribute integer  nbre_chaine_fabrication;
   referentiel part :
      attribute string reference_professionnelle;
}

class  Produit_grand_public_fabrication :Produit_fabrication (extent produits_grand_public_fabrication
                                        key reference_grand_public)
{
  viewpoint part :
      attribute integer  duree_fabrication;
  referentiel part :
     attribute string reference_grand_public;
}

class  Entrepot_fabrication (extent entrepots_fabrication  Key nom)
     {
   viewpoint part :
      attribute integer  distence_usine;
  referentiel part :
     attribute integer capacite;
     }
```

Bibliographie

[1]. Abiteboul, S., Hull, R., & Vianu, V. (1995). Foundations of Databases. Eddition-Wesley, Reading, Mass.

[2]. Abiteboul. S & Bonner. A. (Juin 1991). Objects and views. Proc. International Conference on Management of Data. (pp. 238-247)., Denver, Colorado.

[3]. Adiba, M. (Juillet, 1980). Derived relation : a unified mechanism for views, snapshots and distributed data. In IBM Research Report, California, USA.

[4]. Albano, A., Bergamini, R., Ghelli, G., & Orsini, R. (1993). An Object Data Model with Roles. Proceedings of the 19th International Conference on Very Large Data Bases, (pp. 39-51), Dublin, Ireland. Morgan Kaufmann. ISBN1-55860-152-X.

[5]. Albano, A., Ghelli, G., & Orsini, R. (1995). Fibonacci: A Programming Language for Object Databases. The VLDB Journal (Very Large Data Bases), 4(3), (pp. 403-444) . Special Issue on Persistent Object Systems.

[6]. ANSI/X3/SPARC (1978). Framework Report Study Group on Database Management Systems. Information Systems, 3(3). (pp.173-191).

[7]. Bancilhon, F, Delobel. C & Kanellakis. P. (1992). Building an Object-Oriented Database System.The story of O2. Morgan Kaufman, San Mateo, California.

[8]. Bachaman, C.W. & Daya M. (1977). The role concept in Data models. In Proceeding of the Third International Conference on Very Large DataBases, (1977),pp: 464-476, Tokyo, Japan.

[9]. Bancilhon, F., Delobel, C. & Kanellakis, P. (1992). Building an Object-Oriented Database System.The story of O2. Morgan Kaufman, San Mateo, California.

[10]. Bardou D. (1998). Etude de langages à prototypes, du mécanisme de délégation et de son rapport à la notion de point de vues. Thèse de doctorat en Informatique, LIRMM, université de Montpellier 2.

[11]. Bauzer Medeiros, C., & Pfeffer, P. (1991). Object Integrity Using Rules. Proceedings of ECOOP'91, P. America (Editor), Springer-Verlag, (pp. 219-230), Geneva, Switzerland.

[12]. Benchikha, F. & Boufaida, M. (October 1998). Un modèle conceptuel pour une représentation multiple et évolutive des connaissances. Proc. The 4th African Conference on Research in Computer Science (pp. 793-804). Dakar, Sénégal.

[13]. Benchikha, F., Boufaida, M. & Seinturier, L. (2001). The integration of the Viewpoint Mechanism in Federated Databases. Proc. ACM, SAC'01 (pp. 280-284). Las Vegas.

[14]. Benchikha, F., Boufaida, M. (2005), Extending Object Oriented Databases to Support the Viewpoint Mechanism. Prtoceeding of ICEIS'05, (pp. 273-278), 25-28 May 2005, Miami/USA.

[15]. Benchikha, F., Boufaida, M. (2005). Viewpoints : A Framwork for Object Oriented Databases Modelling and Distribution. Data Science Journal, Vol 4 , (pp. 92-107),

[16]. Bertino, E. (March 1992). A View Mechanism for Object-Oriented Databases. Proc. The 3rd International Conference on EDTB'92 (pp. 136-151). Australia.

[17].Bertino, E., & Martino, L., (1993). Object-Oriented Database Systems. Concepts and Architectures, International Computer Science Series, Addison-Wesley.

[18].Bobrow, D.G. Winograd, T. (1977). An Overview of KRL, a Knowledge Representation Language. Cognitive Science, Vol. 1.

[19].Bobrow D.G., & Stefik M.(1983). The LOOPS Manual : a Data and Object-Oriented Programming System for Interlisp, Knowledge-Based VLSI Design Group, Memo KB-VLSI-81, Xerox PARC, Palo Alto, California.

[20].Breitbart, Y. & Silberschatz, A. (June 1988). Multidatabase update issues. Proc. SIGMOD International Conference on Management of Data (pp. 135-142).

[21].Bouaziz, T., (1995). Classification des contraintes d'intégrité dans les bases de données orientées-objet. Ingénierie des systèmes d'information, vol 3 n°6, Hermès (Editeur), (pp.713-737).

[22].Bukhres, O.A. & Elmagarmid, A.K. (1996). Object-Oriented Multidatabase Systems. Prentice-Hall, Englewood Cliffs, NJ.

[23].Cardelli, L. & Wegner, P. (1985). On understanding types, data abstraction, and polymorphism. ACM Computer Survey. 17(4), (pp. 471-522).

[24].Carn, N. (October 1992). Représentation Orientée Objet de Système Opérationnel avec application au domaine spacial. INP Thesis, Toulouse, France.

[25].Carré B., (1989). Méthodologie orientée objet pour la représentation des connaissances, concepts de points de vue, de représentation multiple et évolutive d'objets", Thèse du LIFL, 1989.

[26].Carré B., Dekker, L., & Geib J.M., (1990). Multiple and Evolutive Représentation in the ROME language. A*ctes de "TOOLS'90"*, (pp. 101-109).

[27].Cattel, R.G.G., Atwood, T., Dubl, J., Ferran, G., Loomis, M., & Wade, D. (1993) The Object Database Standard: ODMG-93. Morgan Kaufmann.

[28].Cattel, R.G.G., Barry, D., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H., & Wade, D. (&1997) The Object Database Standard: ODMG-2.0. Morgan Kaufmann.

[29].Charrel, P. J., Galaretta, D., Hanachi, C., Rothenburger, B. (October 1993) Multiple Viewpoints for the Development of Complex Software. IEEE International Conference on Systems, Man and Cybernetics (pp. 556-561).

[30].Coulondre, S. (2000). Samovar : un modèle pour les objets persistants avec rôles. Thèse de Doctorat d'Université, Université Montpellier II, Montpellier, France.

[31].Cueignet, X., & Lextrait, V. (1992). Génération de serveurs de vues. Thèse de l'université de SophiaAntipolis.

[32].Debrauwer, L. (1998) Des vues aux contextes pour la structuration fonctionnelle de bases de données à objets en CROME. Doctorat thesis, University of sciences and technologies, Lile, France.

[33].Dekker L., & Carré B., (Juin 1992). Multiple and dynamic representation of frames with points of view in FROME. Ac*tes de "Représentation Par Objets"*, La Grande Motte, (pp. 97-111).

[34].Dekker, L. (1994). FROME: Représentation Multiple et Classification d'Objets avec Points de Vues". Thèse de Doctorat, University of sciences and technologies, Lile, France.

[35].Delannoy, X., (1994). La cohérence dans les bases de données. Rapport de recherche RR 936 I, IMAG.

[36].Delobel, C,. & Adiba. M. (1982). Bases de Données et Systèmes relationnels. CNRS. Paris, Dunod Edition.

[37].Delobel, C,. Lécluse. C., & Richard, P. (1991). Bases de données : des systèmes relationnels aux systèmes à objets. Paris, InterEditions édition.

[38].Dos Santos. C.S. (Novembre 1995). Un mécanisme de vues dans les systèmes de gestion de bases de données. Thèse de Doctorat d'Université, Université Paris Sud, Orsay, France.

[39].Finkelstein A., Kramer J., & Goedicke M., (Décembre 1990). Viewpoint Oriented Software Development. Proceedings of Software Engineering and Applications Conference, (p. 337-351), Toulouse.

[40].Finkelstein A., Gabbay D., Hunter A., Kramer J., & Nuseibeh B., (Septembre 1993). Inconsistency Handling in Multi-Perspective Specifications. Aactes de conférence ESEC'93, Garmish-Paternkirchen (D), (pp. 84-99).

[41].Fishman D.H. & OTHERS, Iris: An Object-Oriented DBMS, (1987), ACM Transactions on Office Automation Systems, 5(1).

[42].Gergatsoulis, M., Stavrakas, Y., Karteris, D., Mouzaki A., & Sterpis, D. (2001) A Web-Based System for Handling Multidimentional Information through MXML. Lecture Notes in Computer Science (LNCS 2151) (pp. 352-365), Springer-Verlag.

[43].Goldstein, I. P., & Bobrow, D. G. (1980). Extending Object Oriented Programming in Smalltalk. Actes de "Lisp Conference Standford", (pp. 75-81).

[44].Gottlob, G., Schrefl, M., & R ock, B. (1996). Extending Object-Oriented Systems with Roles. ACM Transactions on Information Systems (TOIS), 14(3). (pp. 268-296).

[45].Halevy, A. (2000) Logic-based techniques in data integration. Proc. Logic Based Artificial Intelligence.

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

[47].Heiler. S & Zdonik. S. (Septembre 1988). Views, Data Abstraction and Inheritance in the FUGUE Data Model. Advances in Object-Oriented Database Systems, Proc. 2nd International Workshop on Object-Oriented Database Systems (pp. 225-241). Lecture Notes in Computer Science (LNCS 334).

[48].Heimbigner, D., & McLeod, D. (July 1985) A Federated Architecture for Information Systems. ACM Transactions on office Information Systems 3(3), 253-278.

[49].Kiczales G., Lampng J., Mendhekar A., Maeda C., & Lopes C.V.(Juin 1997). "Aspect-Oriented Programming", Proceddings of the European Conference on Object-Oriented Programming (ECOOP), Finland, Springer-Verlag LNCS 1241.

[50].Kiczales G., Hilsdale E., Hugunin J., Kersten M., Palm J., & Griswold W.G., (2001). "An Overview of AspectJ", Proceeding of ECOOP'01, Springer Verang LNCS2072.

[51].Kriouile, A., (Octobre 1995) VBOOM, une méthode orientée objet d'analyse et de conception par points de vue. Doctorat thesis, University of Mohamed V, Rabat, Maroc.

[52].Kuno, H. A., Ra, Y.-G., & Rundensteiner, E. A. (1995). The Object-Slicing Technique: A Flexible Object Representation and Its Evaluation. Rapport Technique CSE-TR-241-95, University of Michigan, Ann Arbor, Michigan, USA.

[53].Kuno, H. A. & Rundensteiner, E. A. (1996). The MultiView OODB View System: Design and Implementation. Theory and Practice of Object Sytems (TAPOS), 2(3). (pp. 202-225). Special Issue on Subjectivity in Object-Oriented Systems.

[54].Lemoigne, J.L. (1990). La modélisation des systèmes complexes, Dunod Edition.

[55].Lieberman, H. (1986).Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems. Proceedings of the 1st Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'86), (pp. 214-223), Portland Oregon, USA. Published as ACM SIGPLAN Notices 21(11).

[56].Marino, O. (1993) Raisonnement classificatoire dans une représentation à objets multi-points de vue. Doctorat thesis, University of Joseph-Fourier, Grenoble, France.

[57].Manolescu, I., Florescu D., & Kossmann, D. (2001) Answering XML Queries over Heterogeneous Data Sources. Proc. The 27th VLDB Conference, Roma, Italy.

[58].Menzies, T., Easterbrook, S., Nuseibeh, B., & Waugh, S. (June 7-11, 1999) An Empirical investigation of multiple viewpoint reasoning in requirements engineering. Proc. The fourth International Symposium on Requirements Engineering (RE'99), Limerick, Ireland.

[59].Mili H., & Dargham J., (1997).View Programming in C++: A co-reference based approach. Rapport technique, Département d'Informatique, Université du Québec à Montréal, Canada.

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

[61].Mili H, Mcheick H., Dargham J., & Dalloul S. (2001). Distribution d'objets avec vues", Revue L'Objet-7/2001, LMO'2001, (pp. 27-44).

[62].Mili H., Mcheick H., & Sadou S., (2002). CorbaViews – Distributing Objects that Support Several Functional Aspects. in Journal of Object Technology 1(3), Special issue : TOOLS USA 2002 proceedings, (pp. 207-229).

[63].Minsky, M. (1975). Framework for Representing Knowledge. In the Psychology of Computer Vision. P.H. Winston (éd.), M*cGrawHill, New York, Chapitre 6, (pp. 156-189).

[64].Morejon, J. (1994). Merise: Vers une modélisation orientée objet. Collection Ingénierie des Systèmes d'Information, Les Editions d'Organisation.

[65].Motshnig-Pitrik R., Schett M., (2003). Customizing web-based systems with object-oriented views. Proceedings of 5th International Conference on Enterprise Information Systems ICEIS'03, Angers. (pp. 23-26).

[66].Nassar. M,. (2005). Analyse et conception par points de vue : le profil VUML. Thèse de Doctorat de l4institut National Polytechnique de Toulouse. France

[67].Nassif, Y. Qiu & Zhu. J,. (1991). Extending The Object-Oriented Paradigm to Support Relationships and Constraints, in Object-Oriented Databases: analysis, Design & Construction. (DS-4), R.A. Meersman, W. Kent and S. Khosla (Editors), IFIP, Elsevier Science Publishers.

[68].Nguyen, G. T., & Rieu, D. (March 1991) Database Issues in Object-Oriented Design. Proc. The 4th International Conference TOOLS (pp. 73-86), Paris.

[69].Nuseibeh, B., Finkelstein, A., &Kramer, J. (1996). Method Engineering for Mutli-Perspective Software Development. Information and Software Technology Journal, 38(4), (pp. 267-274), Elsevier Science B. V.

[70].Papazoglou, P. M. & Kramer, B. J. (1997). A Database Model for Object Dynamics. The VLDB Journal, 6(2). (pp. 73-96).

[71].Pernici, B. (1990) Objects with roles. Proc. Office Information Systems (pp. 205 215), ACM.

[72].Pons, A. (June 1992) Formalisation logique d'un mécanisme d'héritage crédule avec points de vue. Proc. Actes des Journées RPO, Edition EC2 (pp. 73-85) , La Grande Motte.

[73].Rathke, C., & Redmiles, D.F.(March 1993) Multiple Representation Perspectives for Supporting Explanationin Ontext. Technical Report CU-CS-645-93, University of Colorado, Department of Computer Science, Boulder, Colorado.

[74].Richardson, J. & Schwarz, P. (1991). Aspects: Extending Objects to Support Multiple, Independent Roles. Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, (pp. 298-307), Denver, Colorado, USA. Published as SIGMOD Record 20(2).

[75].Rieu, D., Nguyen, G. T., Culet, A., Escamilla, J., & Djeraba, C. (1991) Instanciation Multiple et Classification d'Objets. VII^{èmes} Journées Bases de Données Avancées, Lyon.

[76].Ross, D., & Schaman, K.E. (1977) Structured Analysis for Requirements Definition. IEEE Transactions 3(1), 6-15.

[77].Rundensteiner, E. (1992) Multiview: a Methodology for supporting Multiple Views in Object-oriented Databases. Proc. The 18th VLDB Conference. (pp. 187-198), Canada.

[78].Rundensteiner, E. A. (1993).Tools for View Generation in Object-Oriented Databases. Dans Bhargava, B. K., Finin, T. W., et Yesha, Y., _editeurs, Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM'93), (pp.635{644), Washington, DC, USA. ACM Press.

[79].Scholl, M. H., Laasch, C., Rich, C., Schek, H.-J., & Tresch, M. (1992). The COCOON Object Model. Rapport Technique 192, Department of Computer Science, ETH Zurich, Zurich, Switzerland.

[80].Scholl, M. H., Laasch, C., & Tresch, M. (1991). Upda-table Views in Object-Oriented Databases. Dans Delobel, C., Kifer, M., et Masunaga, Y., _editeurs, Proc. of the 2nd Deductive and Object-Oriented Databases International Conference (DOOD'91), volume 566 de Lecture Notes in Computer Science, (pp. 189-207), Munich, Germany. Springer-Verlag.

[81].Sheth, A. & Larson, J. (September 1990) Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous databases. ACM Computing Surveys 22(3), 183-236.

[82].Shilling, J.J. & Sweeney, P.F. (1989) Three Steps to Views: Extending the Object-Oriented Paradigm. Proc. OOPSLA'89 (pp. 353-361).

[83].Sciore, E. (April 1989) Object Specialisation. ACM Trans. Information Systems 7(2),103-122.

[84].Sowa, J. F., (1984). Conceptual Structures. Information Precessing in Mind and Machine, Addition-Wesley publishing.

[85].Stavrakas, Y. Gergatsoulis, M. & Mitakos, T. (2000) Representing context-dependent information using Multidimentional XML. Proc. The 4th European Conference

ECDL'2000, Lecture Notes in Computer Science (LNCS 1923) (pp. 368-371), Springer-Verlag .

[86].Stavrakas, Y. Gergatsoulis, M. & Rondogiannis, P. (2000) Multidimentional XML. Proc. The Third International Workshop (DCW'2000), Lecture Notes in Computer Science (LNCS 1830) (pp. 100-109), Springer-Verlag .

[87].Stefik, M., & Bobrow, D. G. (1985). Object-oriented programming: themes and variations. A.I. magazine, 6(4), (pp. 40-62).

[88].Tanaka. K, Yoshikawa. M., & Ishihara. K. (Février 1988). Schema Virtualization in Object-Oriented Databases. IEEE Data Engineering (pp. 23-30), Los angeles, USA.

[89].Temmerman. N. (1984). Les problèmes de vues et d'intégrité dans les bases de données relationnelles et dans le SGBD relationnel SABRE. Thèse de Doctorat de l'Université Pierre et Marie Curie. France.

[90].Tresch, M. & Scholl, M. H. (1993). Schema Transformation without Database Reorganization. ACM SIGMOD Record, 22(1), (pp.21-27).

[91].Vanwormhoudt G., (1999). CROME : un cadre de programmation par objets structurés en contextes. Thèse de Doctorat en Informatique, LIFL, université des sciences et technologies de Lille.

[92].Vossen, G. (1994). Formalization of OODB Models. Proceedings of the 1st Workshop KRDB'94, Germany.

[93].Wolinski, F., & Perrot, J. F. (1991). Representation of complex objects: Multiple Facets with Part-Whole Hierachies. Actes de "ECOOP'91", (pp. 288-306), Genève.