Méta-modélisation, mise en oeuvre et optimisation des performances des systèmes décisionnels

Khadija Letrache

N° d'ordre : 01/2019/CED

FACULTE DES SCIENCES ET
TECHNIQUES DE MOHAMMEDIA
UNIVERSITE HASSAN II DE CASABLANCA

Université Hassan II de Casablanca

Thèse de Doctorat

Présentée par :

Mme Khadija LETRACHE

Spécialité :
Informatique

Sujet de la thèse :

Méta-modélisation, mise en œuvre et optimisation des performances des systèmes décisionnels

Thèse présentée et soutenue à Mohammedia le 19/01/2019 devant le jury composé de :

Nom et Prénom	Grade	Etablissement	Qualité
Mohamed YOUSSFI	PES	ENSET-Université Hassan II de Casablanca	Président
Rachid OULAD HAJ THAMI	PES	ENSIAS-Université Mohammed V de Rabat	Rapporteur
Mostafa BELLAFKIH	PES	INPT-Rabat	Rapporteur
Mohamed KISSI	PES	FSTM- Université Hassan II de Casablanca	Rapporteur
Hatim HAFIDDI	PH	INPT-Rabat	Examinateur
Omar EL BEGGAR	PA	FSTM- Université Hassan II de Casablanca	Co-encadrant
Mohammed RAMDANI	PES	FSTM- Université Hassan II de Casablanca	Directeur de thèse

Etablissement : Faculté des Sciences et Techniques - Mohammedia
CEDoc : Sciences, Techniques, Ingénierie et Développement Durable
Nom du laboratoire : Laboratoire Informatique de Mohammedia

A mon très cher mari

A ma très chère mère

A mes très chers trésors

A ma famille

A mes amis

Pour votre soutien, votre amour et …. Votre présence

Remerciements

Je tiens à exprimer ma profonde gratitude au Professeur **Mohammed RAMDANI**, directeur de cette thèse, pour m'avoir offert l'opportunité d'entamer cette thèse et faire partie de son équipe. Je le remercie également pour sa disponibilité, ses conseils précieux et pour tout le savoir et les riches expériences qu'il n'hésite jamais à partager avec tous ses étudiants. Enfin, je voudrais le saluer pour son intérêt soutenu pour la recherche scientifique.

J'adresse mes remerciements les plus sincères au Professeur **Omar EL BEGGAR**, co-encadrant de cette thèse, pour sa disponibilité et son suivi, depuis le premier jour et tout au long de ces années de thèse. Je le remercie pour sa rigueur, son esprit d'équipe et ses encouragements. Je le remercie également pour ses conseils et ses remarques constructives qui ont grandement contribué à améliorer la qualité de mes travaux.

Je tiens à mentionner le grand honneur que m'ont fait Professeur **Mostafa BELLAFKIH**, Professeur **Rachid OULAD HAJ THAMI** et Professeur **Mohamed KISSI** en acceptant d'être rapporteurs de mon travail. Je les remercie pour l'intérêt porté à ma thèse et pour l'honneur qu'ils me font en participant au jury.

Je tiens à mentionner également le plaisir et l'honneur que m'a fait Professeur **Mohamed YOUSSFI** d'avoir accepter de présider mon jury de thèse.

Je tiens à mentionner également le plaisir et l'honneur que m'a fait Professeur **Hatim HAFIDI** d'avoir accepter d'être membre de mon jury de thèse.

Résumé

Les travaux présentés dans cette thèse se situent dans le cadre de la modélisation, l'implémentation et l'optimisation des systèmes décisionnels. En fait, vu leur caractère évolutif et complexe, les systèmes décisionnels requièrent un coût et un délai de mise en place et de maintenance très élevés. D'autre part, bien que les systèmes décisionnels soient dédiés à l'analyse et le stockage de grands volumes de données et malgré l'utilisation d'outils décisionnels sophistiqués, les performances des systèmes décisionnels peuvent connaître des dégradations importantes au fil du temps. S'ajoute également, la gestion de l'espace de stockage et des stratégies de rafraîchissement qui devienne une tâche fastidieuse.

Ainsi, la première contribution des travaux de cette thèse est l'élaboration d'une approche dirigée par les modèles qui fournit une démarche et des outils permettant de concevoir et modéliser les systèmes décisionnels et d'automatiser leur cycle de développement. L'approche proposée vise ainsi à réduire le coût et le délai de développement et de maintenance. Elle vise également à garantir une indépendance vis-à-vis des plateformes utilisées et à faciliter la migration d'une plateforme vers une autre.

Notre deuxième contribution porte sur l'optimisation des cubes OLAP. A cet égard, nous proposons une approche de partitionnement horizontal basée sur les règles d'association. L'approche proposée permet, à la fois d'améliorer les performances des cubes OLAP et de faciliter leur maintenance.

Abstract

The works presented in this thesis relate to decision-support systems modeling, implementation and optimization. In fact, because of their scalability and complexity, decision-support systems require a very high cost and time to be implemented and maintained. On the other hand, although they are dedicated to analyze and store huge amount of data and despite the use of sophisticated OLAP tools, decision-support systems can experience significant performance degradation over time. Furthermore, storage and refresh strategies management becomes a tedious task.

Thus, the first contribution of this thesis is to elaborate a model-driven approach and provide tools for designing and modeling decision-support systems and also automating their development lifecycle. The proposed approach aims to reduce time and cost needed for the project development and maintenance, as well as ensuring its independency from the used platforms and facilitating the migration from a platform to another.

Our second contribution deals with OLAP cubes optimization. In this respect, we propose a horizontal partitioning approach based on the association rules. The proposed approach allows improving OLAP cubes performance and facilitating their maintenance.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

ASSL	Analysis Services Scripting Language
ATL	Atlas Transformation Language
BI	Business Intelligence
BPM	Business Process Management
BPMN	Business Process Model and Notation
CIF	Corporate Information Factory
CIM	Computation Independent Model
CWM	Common Warehouse Model
DFM	Dimensional Fact Model
DSL	Domain Specific Language
DSS	Decision Support System
DWB	Data Warehouse Bus
DM	Data Mart
DW	Data Warehouse
ED	Entrepôt de Données
EMF	Eclipse Modeling Framework
ER	Early Requirement
E/S	Entrée/Sortie
ETL	Extract Transform and Load
GA	Genetic Algorithm
GAF	Genetic Algorithm Framework
GDI	Goal Decision Information
GMF	Graphical Modeling Framework
GRT	Goal Refinement Tree
HC	Hill Climbing
HOLAP	Hybrid OLAP
JAXP	Java API for XML Processing
KPI	Key Performance Indicator
LR	Late Requirement
M2M	Model to Model
M2T	Model to Text
MDA	Model Driven Architecture
MDC	Meta Data Coalition
MDE	Model Driven Engineering
MDX	MultiDimensional eXpressions
MOF	Meta Object Facility
MOLAP	Multidimensional OLAP
OCL	Object Constraint Language
ODS	Operational Data Source

OIM	Open Information Model
OLAP	OnLine Analytical Processing
OLTP	OnLine Transaction Processing
OMDA	OLAP Model Driven Architecture
OMG	Object Management Group
OPAR	OLAP Partitioning based on Association Rules
OWL	Ontology Web Language
OWL-DL	Ontology Web Language Description Logics
PDM	Platform Description Model
PIM	Platform Independent Model
PSM	Platform Specific Model
PUM	Predicates Usage Matrix
QS	Qualité de service
QVT	Query/View/Transformation
ROLAP	Relational OLAP
SA	Simulated Annealing
SD	Strategic Dependency
SGBD	Système de Gestion des Bases de Données
SGAP	Strategic Goal Analysis Process
SLDC	Software Developpement Life Cycle
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SR	Strategic Rational
SSAS	SQL Server Analysis Services
SSQL	Specification SQL
TPC-DS	Transaction Processing Performance Council Decision Support
UML	Unified Modeling Language
XML	eXtensible Markup Language
XMLA	XML for Analysis
WSBPEL	Web Service Business Process Execution Language

Introduction

Contexte et motivations

De nos jours, les organisations font, plus que jamais, face à la compétitivité et la difficulté de promouvoir leur productivité et atteindre leurs objectifs. Ceci est dû en particulier à la crise économique qu'a connue le monde durant cette dernière décennie (Saber, 2011). De ce fait, l'informatisation, l'analyse et le suivi en temps réel des résultats sont devenus indispensables pour identifier les opportunités et minimiser les risques, face à une incroyable croissance technologique, qui fait multiplier les sources d'informations et exploser le volume des données. D'où le rôle de l'informatique décisionnelle.

En effet, l'informatique décisionnelle (ou BI pour Business Intelligence) comprend un ensemble de méthodologies, de processus, d'architectures et de technologies capables de transformer les données brutes en informations pertinentes et utiles pour la prise de décision (Vaisman, 2014). Les données sont alors collectées, transformées et stockées dans un entrepôt de données ou Data warehouse (DW). Par la suite, ces données sont analysées via des outils d'analyse, dont le plus typique est l'OLAP (Online Analytical Processing).

Toutefois, après près de 30 ans d'existence et de développement des systèmes décisionnels, le constat général, tiré des retours d'expériences, affirme la complexité des projets décisionnels et leur ampleur en termes de coût et de délai de mise en place (Prakash et al, 2018; Saber, 2011). Outre ce coût financier, pouvant atteindre le 1 million de dollars durant la première année (Hayen et al, 2007), les projets décisionnels sont également coûteux en termes de ressources (Hughes, 2012). D'autre part, les projets décisionnels génèrent un coût de maintenance élevé, pouvant atteindre les 90% du coût global du projet (Prakash et al, 2018), à cause en particulier de leur caractère évolutif. En effet, en plus de la difficulté de gestion de l'espace de stockage dû à

l'énorme taille des entrepôts de données et des bases de données OLAP, s'ajoute également la difficulté de mise à jour de ces derniers et la dégradation des performances que cela entraine.
Ce sont également des projets à haut risque d'échec, et dont le résultat final ne correspond pas, généralement, aux attentes des utilisateurs (Prakash et al, 2018). Une enquête publiée par Ericson affirme que 65% des projets décisionnels, et dont le coût est d'environ 12 millions de dollars, échouent (Ericson, 2006). Les causes de cet échec sont reliées, semblablement, à toutes les phases du cycle de vie du système, depuis la définition des besoins jusqu'à l'implémentation finale (Prakash et al, 2018). En effet, d'un côté les opérations ne savent pas formuler leur besoin, parce qu'ils n'ont pas d'expériences avec les systèmes décisionnels. D'autre part, la conception du système s'étale sur une longue durée et se trouve perdue entre des définitions contradictoires des données (Hughes, 2012).
Dans ce sens, parmi les solutions pouvant réduire les ressources, coût et délai de développement et de maintenance des projets décisionnels est l'automatisation du processus de développement.
Outre cela, plusieurs actions peuvent être menées après la mise en production du système décisionnel pour améliorer d'un côté ses performances mais aussi réduire son coût de maintenance en facilitant la gestion des mises à jour et la gestion de stockage.

L'architecture dirigée par les modèles (MDA) dans l'élaboration des systèmes décisionnels

L'une des solutions ambitieuses d'automatisation du cycle de développement des systèmes décisionnels est l'architecture dirigée par les modèles ou la MDA (pour Model Driven Architecture).
Cette dernière, proposée par l'OMG (Object Management Group) en 2000, est une variante de l'ingénierie dirigée par les modèles (Bézivin et al, 2004), qui se réfère à l'utilisation systématique des modèles comme des éléments centraux tout au long du cycle de vie du développement logiciel, donnant naissance au nouveau paradigme « Tout est modèle » au lieu du paradigme « Tout est objet ».
La MDA, dont le principe est « Model once, generate everywhere » (Bézivin et al, 2003), permet d'une part l'automatisation du processus de développement mais aussi la portabilité et l'interopérabilité des applications et ainsi, de faire face aux verrous et aux changements technologiques. Par conséquent, les développeurs et concepteurs BI peuvent se concentrer sur les aspects métier au lieu des aspects techniques, et de cette manière, livrer un produit qui répond aux besoins des utilisateurs finaux. En outre, la MDA permet de réduire le coût de développement et de maintenance des projets grâce à la réutilisabilité du processus de transformation et de génération automatique des modèles et du code d'implémentation.

Cependant, malgré la multitude de travaux ayant adopté cette architecture dans l'élaboration des systèmes décisionnels (Lujan-Mora et al, 2006; Mazon et al, 2008; Blanco et al, 2015), on constate l'absence d'une approche complète.

En effet, dans la phase de spécification des besoins utilisateurs, la description des sources de données dans toutes les méthodes proposées, n'est pas intégrée dans les spécifications des besoins. La réconciliation avec les sources de données se fait alors en postériori, ce qui peut entraîner une révision des spécifications après leur validation. De plus, les méthodes largement utilisées comme iStar (Mazon et al, 2007) et GRT (Goal Refinement Tree) (Zepeda et al, 2010) deviennent complexes et illisibles une fois les besoins des utilisateurs s'élargissent (Maté et al, 2014). Ajoutant aussi, l'absence de mécanisme de contrôle des règles de forme. D'autre part, on constate au niveau de la phase de modélisation logique et physique, que toutes les approches proposées se sont focalisées sur le modèle relationnel du DW, en générant son schéma relationnel et le code SQL nécessaire à la création de ses tables, au moment où le modèle OLAP est négligé.

Les techniques d'optimisation des systèmes décisionnels

Bien que les systèmes décisionnels soient dédiés à l'analyse et le stockage de grands volumes de données et malgré l'utilisation d'outils et de langages BI sophistiqués, les performances des systèmes décisionnels peuvent subir des dégradations importantes, dues principalement, à l'accroissement continu du volume de données. Cette dégradation peut concerner le temps de réponse des requêtes ou le temps de rafraîchissement, appelé temps de traitement. Outre cela, les systèmes décisionnels peuvent connaître des problèmes de gestion de l'espace de stockage, en particulier au niveau des bases OLAP. Tout cela contribue à l'augmentation du coût de maintenance du projet.

Pour pallier à cela, plusieurs techniques d'optimisation sont proposées dans la littérature et par les commerciaux. Le défi est, cependant, de choisir la technique la plus appropriée ainsi que la meilleure configuration pour cette dernière.

Parmi ces techniques, on distingue deux catégories, à savoir, les techniques redondantes et les techniques non redondantes.

La première catégorie, incluant les vues matérialisées et les index, est largement utilisée par les éditeurs des SGBD et OLAP. Cependant, ces techniques requièrent un espace de stockage additionnel, en plus des contraintes de sélection et de mise à jour. La deuxième catégorie comprend le partitionnement, représentant la méthode d'optimisation la plus recommandée dans les systèmes décisionnels (Kimball, 2013; Vaisman, 2014), grâce à ses nombreux avantages.

Inmon considère le partitionnement comme la deuxième problématique importante à traiter après celle de la granularité (Inmon, 2005).

Il existe en fait deux types de partitionnement; vertical et horizontal. Une partition relationnelle verticale contient un sous ensemble des colonnes de sa table d'origine. De même, une partition OLAP verticale contient un sous-ensemble des mesures et dimensions de son cube. Ce qui signifie que la partition résultante, relationnelle ou OLAP, n'a pas la même structure que celle d'origine.

Une partition relationnelle horizontale, quant à elle, possède la même structure que sa table d'origine mais avec moins de lignes. De même, une partition OLAP horizontale possède les mêmes dimensions et mêmes mesures que son cube parent, mais avec moins de données. Pour cette raison, le partitionnement horizontal est le plus accommodé pour les modèles multidimensionnels et OLAP.

En effet, en réduisant le nombre de lignes d'une table ou d'une partition OLAP, le partitionnement horizontal améliore le temps de réponse des requêtes. Il permet aussi de réduire le temps de traitement du cube associé, vu que le nombre d'agrégations à recalculer à chaque mise à jour, est aussi réduit. Le partitionnement permet également le traitement parallèle des requêtes et des partitions. En outre, il permet de gérer l'espace de stockage en créant des partitions dont la taille respecte cette contrainte. Enfin, grâce au partitionnement, on peut définir une stratégie appropriée de mise à jour du système décisionnel.

Compte tenu de tous ses avantages, le partitionnement horizontal est supporté par la majorité des SGBD et framework OLAP. Le défi est, toutefois, de définir la stratégie de partitionnement appropriée.

Parmi les approches de base qui ont été proposées dans la littérature, on trouve:

- L'approche COM_MIN basée sur les prédicats des requêtes utilisateurs (Ceri et al, 1982). Elle définit le schéma de partitionnement comme étant l'ensemble des conjonctions des prédicats simples, appelés minterm, à partir d'un ensemble de prédicats minimal et complet. Cependant, la complexité du problème est, dans ce cas, de l'ordre de 2^n, pour n prédicats, c.à.d 2^n partitions à créer et à gérer.
- L'approche basée sur l'affinité des prédicats: elle consiste à définir une matrice d'affinité des prédicats à partir de la liste des prédicats simples, en considérant, cette fois-ci, la fréquence des requêtes pour tenter de réduire le nombre de solutions.
- L'approche basée sur un modèle de coût: elle consiste à sélectionner parmi un ensemble de solutions initiales, le meilleur schéma de partitionnement en se basant sur un modèle de coût.

En résumé, toutes ces approches concernent les bases de données ou les entrepôts relationnels. De plus, en dépit des solutions proposées pour réduire le nombre de partitions générées, ce dernier reste toujours élevé. Enfin, toutes ces approches ignorent le volume de données et la taille des partitions à créer. Par conséquent, elles augmentent le coût de maintenance et ne contribuent pas la gestion de l'espace de stockage ni à la réduction du temps de rafraîchissement des cubes OLAP.

Contributions de cette thèse

Nos contributions portent sur deux axes principaux: d'une part la modélisation et l'automatisation du processus de développement des systèmes décisionnels et d'une autre part, l'amélioration des performances de ces derniers.

A cet égard, la première contribution de cette thèse est une approche mixte, orientée buts pour l'élaboration des systèmes décisionnels et qui s'inscrit dans le cadre de l'architecture dirigée par les modèles (MDA). Notre proposition OMDA (OLAP Model Driven Architecture), vise à fournir des outils pour assister les décideurs à définir leur besoin en termes d'analyse, à partir de leurs buts stratégiques. Elle vise, également, à automatiser le processus de développement des systèmes décisionnels via l'adoption d'une architecture MDA. Permettant ainsi, de surpasser les verrous technologiques des outils BI et de réduire le coût et le délai de mise en place et de maintenance du projet. Les étapes de notre proposition peuvent être résumées comme suit:

1- Assister les décideurs à définir leurs besoins exprimés sous forme de buts stratégiques. Ceci est réalisé en deux phases. D'abord, définir les exigences de haut niveau, vue sommaire des besoins. Ensuite, décomposer ces derniers pour obtenir les besoins détaillés, dites exigences de bas niveau. Ceci correspond au niveau CIM (Computation Independent Model) de l'architecture MDA.
2- A partir des besoins détaillés, obtenir un modèle conceptuel, dit modèle PIM (Platform Independent Model) du nouveau data warehouse (DW) ou data mart (DM), suivant un ensemble de règles de transformation semi-automatiques.
3- A partir du niveau conceptuel (PIM), générer automatiquement deux modèles spécifiques aux plateformes utilisées, dit modèles PSM (Platform Specific Model) à savoir, le modèle relationnel et le modèle OLAP. Cette phase intègre également l'utilisation d'un modèle décrivant les spécificités de la plateforme OLAP cible, appelé modèle PDM (Platform Description Model), afin de permettre une transformation automatique du PIM au modèle OLAP.

4- Par la suite, générer le code d'implémentation du DW et cube OLAP à partir des deux modèles spécifiques obtenus précédemment.
5- Enfin, dans le cadre de la gouvernance des données, notre approche traite, également, l'exploitation des modèles obtenus (CIM, PIM, PSMs) pour la collecte des métadonnées business et techniques disponibles dans chaque niveau de l'architecture MDA. Ces métadonnées serviront, par la suite, à documenter le système mis en place. Nous intégrons à ce niveau également, les métadonnées extraites du modèle ETL pour tracer la lignée des données.

L'approche proposée inclut la proposition d'un ensemble de métamodèles servant à représenter chaque modèle de l'architecture MDA, ainsi que les règles de transformation requises pour l'obtenir. Enfin, l'approche est validée à travers une étude de cas réel.

La seconde contribution de cette thèse est la proposition d'une nouvelle approche de partitionnement horizontal, appelée OPAR (OLAP Partitioning based on Association Rules), visant à améliorer les performances des cubes OLAP et réduire leur coût de maintenance. L'approche proposée prend en compte les besoins des utilisateurs à travers la fréquence et la criticité de leurs requêtes, ainsi que la régularité des tailles des partitions obtenues pour faciliter la gestion de l'espace de stockage.
L'approche proposée est basée sur les concepts des règles d'association. Elle comprend deux phases principales:

1- L'analyse des requêtes utilisateurs: Le but de cette phase est d'analyser les requêtes des utilisateurs, exprimées en langage multidimensionnel MDX (Multi Dimensional eXpression), pour déduire les itemsets fréquents de prédicats et qui vont constituer l'entrée de l'algorithme de partitionnement. Pour ce faire, nous utilisons l'algorithme apriori. Cette phase inclut également l'intégration d'un facteur de criticité des requêtes pour privilégier les requêtes qui ne sont pas fréquentes mais qui sont critiques, comme celles utilisées par le top management.
2- Le partitionnement: Lors de cette phase, nous partitionnons le cube OLAP, en prenant comme entrée les itemsets fréquents de prédicats déduits de la phase 1. Pour cet effet, nous proposons un nouvel algorithme de partitionnement basé sur les règles d'association. L'algorithme proposé partitionne le cube progressivement, jusqu'à atteindre un seuil minimum. Pour définir ce seuil, notre approche considère la contrainte de taille maximale des partitions.

Enfin, pour valider notre approche OPAR, nous avons mené plusieurs expérimentations sur la base de benchmark TPC-DS. Les tests réalisés concernent, principalement, l'amélioration du temps de réponse des requêtes, l'amélioration du coût d'entrées/sorties, le nombre de partitions obtenues et la régularité de leurs tailles. La comparaison des résultats obtenus avec ceux de l'état de l'art confirme l'efficacité de l'approche proposée.

Plan de la thèse

Le présent mémoire est organisé en cinq chapitres, comme suit :

Le ***chapitre 1*** rappelle les principaux concepts des systèmes décisionnels, notamment les composants du système et les architectures possibles. Nous évoquons, également dans ce chapitre, la modélisation multidimensionnelle ainsi que le cycle de vie de développement des systèmes décisionnels. Enfin, nous présentons les principales démarches de mise en place de ces derniers.

Le ***chapitre 2*** présente l'architecture dirigée par les modèles, ses concepts de base et ses différents modèles. Il expose ensuite les travaux de l'état de l'art relatifs à l'élaboration des systèmes décisionnels dans le cadre de cette architecture. Enfin, nous établissons une étude comparative de ces travaux selon plusieurs critères afin de mettre en évidence les axes de recherche possibles.

Le ***chapitre 3*** présente notre approche MDA d'élaboration des systèmes décisionnels. Ainsi, pour chaque modèle de l'architecture MDA, nous décrivons son métamodèle et les règles de transformation requises pour l'obtenir. Enfin, nous validons notre approche à travers une étude de cas réel.

Le ***chapitre 4*** présente les techniques d'optimisation existantes relatives aux SGBD et entrepôts de données et met l'accent, par la suite, sur la technique de partitionnement horizontal. Nous exposons dans ce chapitre également les travaux de l'état de l'art et nous mettons en évidence les limitations de chaque proposition.

Le ***chapitre 5*** présente notre approche de partitionnement horizontal, notamment ses deux phases principales, l'analyse des requêtes utilisateurs et le partitionnement. Enfin, nous exposons les résultats des expérimentations comparés à ceux de l'état de l'art pour valider notre approche.

En conclusion, nous présentons le bilan de nos contributions ainsi que nos perspectives d'amélioration et de recherche.

Publications académiques :

Revues scientifiques

Letrache, K., El Beggar, O., & Ramdani, M. (2017). The automatic creation of OLAP cube using an MDA approach. Software: Practice and Experience, 47(12), 1887-1903. Wiley & Sons.(Scopus)

El Beggar, O., Letrache, K., & Ramdani, M. (2017). CIM for data warehouse requirements using an UML profile. IET Software, 11(4), 181-194. (Scopus)

Letrache, K., El Beggar, O., & Ramdani, M. (2018). OLAP cube partitioning based on association rules method. Applied Intelligence, 1-15. (Springer)

Conférences Internationales

Letrache, K., El Beggar, O., & Ramdani, M. (2016, October). Modeling and creating KPIs in MDA approach. In Information Science and Technology (CiSt), 2016 4th IEEE International Colloquium on (pp. 222-227). (IEEE).

El Beggar, O., Letrache, K., & Ramdani, M. (2016, October). Towards an MDA-oriented UML profiles for data warehouses design and development. In Intelligent Systems: Theories and Applications (SITA), 2016 11th International Conference on (pp. 1-6). (IEEE).

Letrache, K., El Beggar, O., & Ramdani, M. (2018, May). Comparative Analysis of Our Association Rules Based Approach and a Genetic Approach for OLAP Partitioning. In International Conference on Networked Systems (pp. 391-403). Springer, Cham.

Chapitre 1

Architecture, Modélisation et Elaboration des Systèmes Décisionnels

Résumé: Depuis le début des années 90 et jusqu'à nos jours, les bases de données multidimensionnelles ont su s'imposer dans les milieux professionnels ainsi qu'académiques. En effet, des millions d'entreprises à travers le monde possèdent des systèmes décisionnels (Dataconomy, 2017). Ce qui justifie l'édition et l'amélioration continues des outils décisionnels proposés par les géants informatiques, tels que Microsoft, Oracle et SAS. A côté de cela, plusieurs travaux de recherche sont toujours d'actualité sur les systèmes décisionnels, tant sur la modélisation que sur l'implémentation et l'optimisation. Dans ce chapitre, nous rappellerons brièvement les principaux concepts d'un système décisionnel et nous décrirons les démarches de mise en place de ce dernier.

1.1 Introduction

En raison de l'incroyable développement technologique que connaît le monde depuis le début du 20ème siècle, la taille des données stockées dans les systèmes opérationnels a considérablement augmenté, allant de quelque méga-octets à plusieurs giga-octets, téraoctets et même péta-octets (Kimball, 2013). Ceci résulte alors en une véritable nécessité de disposer d'outils et de processus permettant d'exploiter et de transformer ce grand volume de données en des informations stratégiques, pertinentes et utiles à l'analyse et la prise de décision. Permettant ainsi aux organisations de maintenir le bon déroulement de leurs activités et de faire face à la grande compétitivité. D'où le rôle de l'Informatique Décisionnelle.

En effet, l'Informatique Décisionnelle ou Business Intelligence (BI) comprend un ensemble de méthodologies, de processus, d'architectures et de technologies capables de transformer les données brutes en informations pertinentes et utiles pour la prise de décision (Vaisman, 2014). On parle ainsi de système d'aide à la prise de décision ou DSS (pour Decision Support System). Ce système a pour mission principale, de purger de vastes volumes de données depuis différentes sources, de les transformer et les stocker dans un entrepôt commun, prêtes à être exploitées par des outils d'analyse.

Kimball définit les exigences du système décisionnel (Kimball, 2013), qu'il note DW/BI, comme suit:

- Le système décisionnel doit rendre l'information facilement accessible: cela signifie que le système doit, avant tout, présenter un contenu compréhensible. Les données fournies par le système doivent être intuitives et évidentes, aussi bien pour les développeurs que pour les utilisateurs métier, et la structure des données et leurs labels doivent refléter le raisonnement et le vocabulaire métier. En outre, l'accès à ces données doit être facile et rapide.
- Le système décisionnel doit fournir des informations cohérentes et crédibles: en effet, les données doivent être soigneusement chargées, nettoyées, de qualité assurée, mais surtout livrées aux utilisateurs qu'une fois valables. La cohérence implique également l'utilisation d'étiquettes de données cohérentes. Ainsi, deux mesures portant le même nom doivent signifier la même chose et inversement.
- Le système décisionnel doit s'adapter aux changements: nouveau besoin, nouvelle règle métier, nouvelles données, nouvelles technologies, tous peuvent se produire. Le système décisionnel doit alors être capable d'intégrer ces changements sans invalider les données ou applications existantes et de manière transparente pour l'utilisateur final.

- Le système décisionnel doit fournir les informations au moment opportun: il doit être capable de charger, traiter et présenter les données aux utilisateurs dans un temps négligeable qui leur permet de prendre des décisions au bon moment.
- Le système décisionnel doit être sécurisé: vu le grand volume de données qu'il héberge, provenant de différentes bases de données opérationnelles (RH, Finance, CRM etc.), le système décisionnel doit être capable de contrôler l'accès à ces données afin de garantir la confidentialité de l'organisation.
- Le système décisionnel doit servir de source fiable et digne de confiance pour la prise de décision. Ceci rejoint aussi la deuxième exigence concernant la cohérence des données. La fiabilité du système décisionnel est aussi un élément crucial pour l'acceptation du système et pour gagner la confiance des utilisateurs.
- La communauté business doit accepter le système décisionnel: peu importe les outils ou plateformes utilisés pour concevoir un système élégant, si les utilisateurs n'embrassent pas ce dernier et l'utilisent massivement le projet est considéré en échec.

Ainsi, pour répondre à toutes ces exigences, les concepteurs et développeurs du système décisionnel doivent connaître aussi bien le métier que les outils de développement. Nous pouvons dire alors qu'un développeur BI doit être moitié DBA (Database Administrator), moitié analyste (Kimball, 2013) pour être en mesure de concevoir un support d'analyse et de prise de décision.

Pour résumer, un système décisionnel est un environnement qui comprend plusieurs processus, composants et outils servant à fournir de l'information fiable, sécurisée, facilement accessible, au moment opportun pour prendre des décisions pertinentes et que les décideurs doivent adhérer pour promouvoir leur activité et faire face à la compétitivité.

1.2. Architecture

1.2.1 Composants

Il existe plusieurs architectures possibles pour un système décisionnel, mais qui partagent toutes les mêmes composants que nous pouvons regrouper en trois catégories ou niveaux comme suit (voir Figure 1.1) :

- ***Extraction***: ce niveau est constitué des **sources de données opérationnelles** et qui sont le principal fournisseur des données qui alimentent l'entrepôt de données. Ces sources sont généralement hétérogènes, réparties et contiennent peu d'historique (Kimball, 2011). Grâce aux outils d'extraction et de transformation ETL (Extract Transform and Load), les données sont extraites depuis les sources de données vers la zone de chargement pour être traitées avant leur

insertion dans l'entrepôt de données. Notant que la zone de chargement peut être classée dans la couche de stockage également.

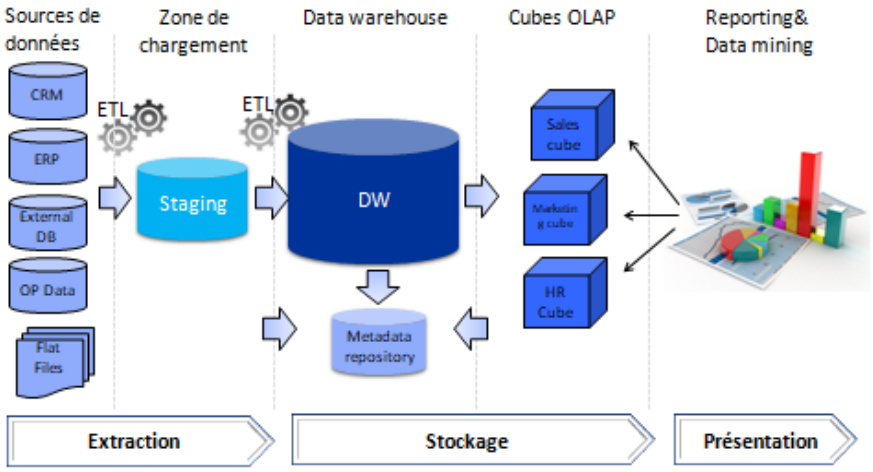


Figure 1.1 Architecture standard des systèmes décisionnels

- ***Stockage***: c'est le principal composant du système décisionnel. Il est constitué principalement de **l'entrepôt de données** (ED) ou data warehouse (DW) défini par Inmon comme étant:
« *Une collection de données intégrées, orientées sujet, non volatiles et à variation dans le temps pour la prise de décision*» (Inmon, 2005).
En effet, un entrepôt de données est orienté thématique ou **sujet** d'analyse relatif aux besoins analytiques exprimés par les décideurs, tels que l'analyse du chiffre d'affaire ou le suivi de la productivité. Il est également **intégré**, du fait qu'il intègre des données provenant de plusieurs sources.
Un entrepôt de données est dit **non volatil** vu qu'il garde les données des systèmes opérationnels pour une longue période. Ces données ne sont ni supprimées ni modifiées, afin de permettre le suivi et l'analyse des changements, contrairement aux bases de données opérationnelles dont le souci primordial est la performance.
Les données de l'entrepôt de données sont **à variation dans le temps,** dites aussi historisées, du fait que le DW garde une trace sur comment les données ont évolué au fil du temps. Par exemple comment les ventes ont évoluées durant les dernières années (Vaisman, 2014). Tandis que les systèmes opérationnels ne gardent que le statut courant des données.
Enfin, l'objectif d'un entrepôt de données est l'analyse de grand volume de données sur un intervalle de temps et dans le but d'aider les décideurs dans la **prise de décision**.
Notant qu'un entrepôt de données peut être organisé en plusieurs magasins de données (data marts), chacun associé à une activité particulière (productivité, marketing, finance etc.).

Le deuxième élément de stockage est la couche **OLAP** (OnLine Analytical Processing), qu'on classifie également comme couche d'analyse ou de présentation. Elle est constituée d'un serveur OLAP, hébergeant des Cubes, et qui est le principal fournisseur des données multidimensionnelles provenant du DW. Contrairement à l'entrepôt de données qui est une base de données relationnelle, pouvant avoir une granularité très fine (Kimball, 2013 ; Ponniah, 2004), un cube OLAP est une structure multidimensionnelle, qui fournit des données agrégées et pré-calculées, appelées mesures, selon différentes perspectives appelées dimensions. Ces agrégations peuvent être stockées selon différents modes de stockage que nous détaillerons dans la section 1.4

Le dernier composant de cette couche est **l'entrepôt des métadonnées** (Metadata repository). Il s'agit d'une base de données (relationnelle ou fichier plat) servant à stocker, classer et gérer les métadonnées (Ponniah, 2004). Une métadonnée est toute information décrivant la structure et la signification des données ainsi que les applications et processus qui les manipulent (MDC-OIM, 1999). C'est une donnée concernant une donnée (Vaisman, 2014).

L'entrepôt des métadonnées est un élément primordial dans la gouvernance des données. Il permet, en effet, de générer la documentation du DW, tracer la lignée des données (data lineage) et surtout garantir la traçabilité des changements dans le système décisionnel. Il est cependant négligé dans la majorité des projets où on consacre plus de temps au développement qu'à la traçabilité.

- ***Présentation:*** est la dernière couche dans l'architecture du système décisionnel. Elle sert à organiser les données afin de permettre aux utilisateurs d'y accéder directement et facilement, via des outils client dédiés, comme les outils de reporting, outils de statistique ou autres applications d'analyse, tels que les outils OLAP ou de data mining. Cette couche, contrairement à ce qu'on pense généralement, doit présenter les données agrégées mais aussi des données détaillées ou granulaires (Kimball, 2013). En fait, l'utilisateur final a toujours besoin de données résumées lui permettant de cibler les phénomènes à analyser, mais pour aller dans le fond de ses analyses, il doit pouvoir visualiser le détail des données ou phénomènes qui l'intéressent. Enfin, cette couche doit présenter les données utilisant le vocabulaire métier sous un format ergonomique, utilisant par exemple des codes couleurs ou des graphes pour faciliter la lisibilité et l'identification d'informations pertinentes.

1.2.2 Les différentes architectures des systèmes décisionnels

a. Data marts dépendants

C'est le type d'architecture le plus connu et le plus utilisé. Parmi ces architectures, celle proposée par Inmon, appelée architecture CIF (*corporate information factory*)(Inmon, 2005). Dans cette architecture (voir Figure 1.2), Inmon propose de créer un data warehouse atomique et normalisé à partir des sources de données opérationnelles. Ensuite, créer des data marts départementaux contenant les données spécifiques à une activité particulière. Contrairement au data warehouse, les données d'un data mart sont dénormalisées, agrégées et dirigées par un besoin exprimé par un département unique. L'architecture CIF intègre aussi un quatrième niveau appelé individuel servant à répondre à des analyses pointues. Il est généralement temporaire et de petite taille.

Cette architecture présente cependant l'inconvénient de cacher le détail aux utilisateurs finaux, du fait que ce détail soit présent dans le deuxième niveau uniquement (Kimball, 2013). En outre, à cause de la normalisation du DW, même en y accédant directement, le temps de réponse des requêtes sera long.

L'architecture CIF présente également plusieurs avantages dont le plus important est qu'elle garantit la conformité des données entre les data marts qui tous, purgent leurs données de la même source, qui est le DW. C'est aussi une architecture flexible dont les data marts peuvent être restructurés et réorganisés à n'importe quel moment, puisque les données sont toujours disponibles dans le DW. Enfin, c'est une architecture facile à mettre à jour (processus ETL simplifié) vu que chaque donnée n'existe que dans un seul endroit.

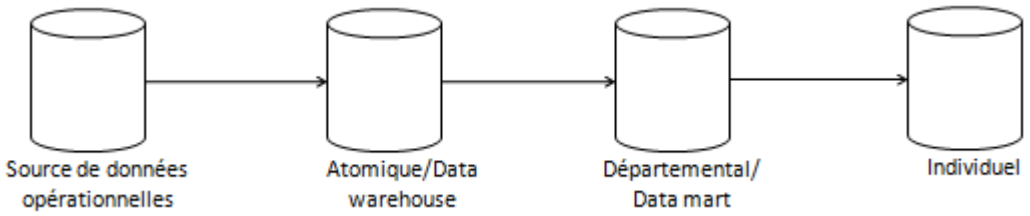


Figure 1.2 Architecture CIF (Inmon, 2005)

Une autre architecture basée sur les data marts dépendants a été proposée par Kimball (Kimball, 2013). Dans cette architecture (Figure 1.3), contrairement à l'architecture CIF, le data warehouse (DW), appelé DW d'entreprise (DWE) est un ensemble de data marts qui ne sont pas forcément normalisés car le but final est de garantir de bonnes performances des requêtes utilisateurs. De plus, les data marts ne sont pas orientés département mais processus métier. Ces data marts sont modélisés avec des schémas multidimensionnels (schéma en étoile) et peuvent contenir des données agrégées ou atomiques. La conformité entre les data marts est assurée par le DWB (Data Warehouse Bus) qui gère l'ensemble des dimensions partagées par les data marts. Il consiste en fait à charger les dimensions via un ETL, indépendamment des data marts, pour ensuite les

distribuées sur tous les data marts concernés. Enfin, les data marts faisant partie de la couche présentation, sont tous directement accessibles par les outils d'analyse ou de reporting.

L'avantage de cette architecture est alors de fournir de bonnes performances. Cependant, elle présente aussi des inconvénients. En effet, contrairement à l'architecture CIF où la conformité des données est assurément et automatiquement garantie, dans l'architecture Kimball, la conformité des dimensions est assurée par le DWB. De plus, à cause de sa dénormalisation, cette architecture est difficile à étendre (Inmon, 2005).

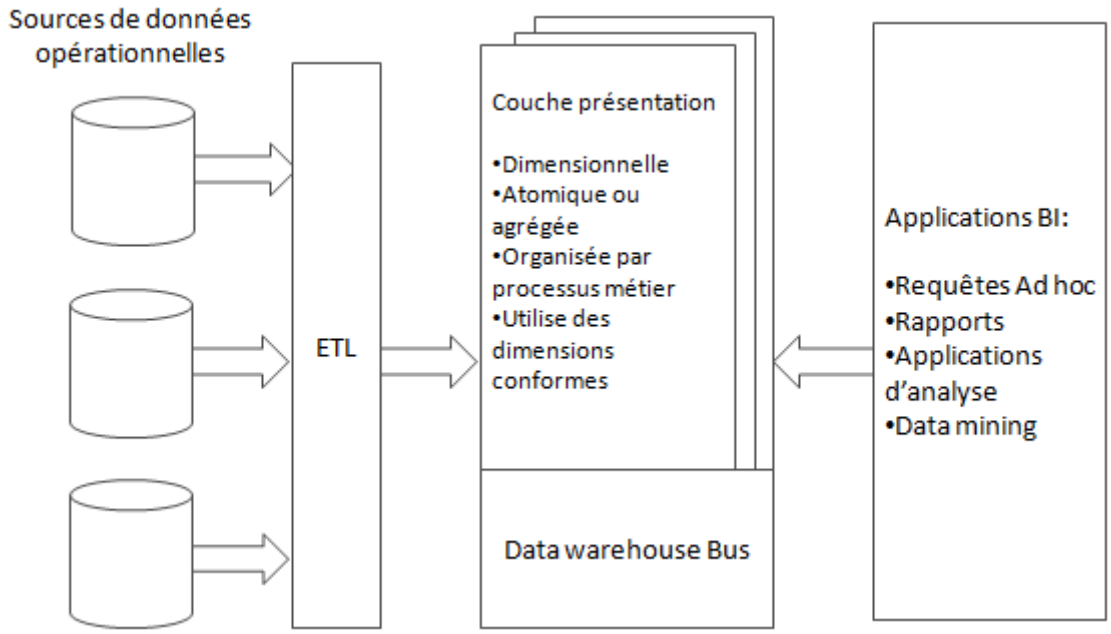


Figure 1.3 Architecture Kimball (Kimball, 2013)

Par ailleurs, Kimball propose une architecture hybride des deux approches présentées précédemment. Pour ce faire, Kimball modifie son architecture en intégrant le niveau DW atomique, normalisé, inaccessible par les utilisateurs finaux et qui constitue la source des data marts. Cette architecture est malheureusement plus coûteuse en termes de coût, d'espace et d'effort de mise en place et de maintenance à cause de la redondance des données atomiques et des différents mouvements des données, mais elle reste un compromis permettant de profiter des avantages des deux architectures, notamment la performance et la flexibilité (Kimball, 2013).

b. Data marts indépendants

Dans cette architecture, chaque data mart est développé séparément (Figure 1.4). Suite à un besoin particulier d'un département, ce dernier, entre en communication directe avec le service IT (Information Technology). Il exprime alors son besoin, ses règles de gestion, son vocabulaire et ses labels et identifie les sources de données requises. Si un autre département a besoin des

mêmes données, provenant du même système, alors il fait lui aussi une demande auprès du service IT et se fait développer un autre data mart spécifique à son besoin et qui se rapproche de très près au premier data mart développé. Les deux, par contre, peuvent avoir des résultats contradictoires, vu que les règles de gestion ne sont pas les mêmes, ainsi que les étiquettes des métriques utilisées. Ainsi, les data marts développés indépendamment ne peuvent être ni regroupés ni partagés, en plus de l'espace de stockage qu'ils requièrent à cause de la redondance des données. Il est donc fortement recommandé d'éviter ce type d'architecture malgré sa simplicité (Kimball, 2013).

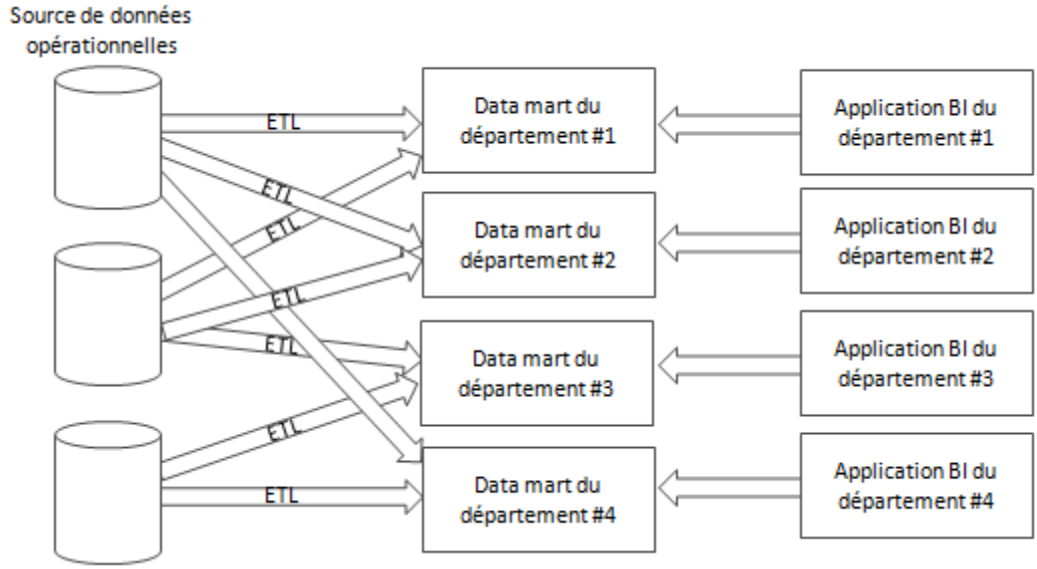


Figure 1.4 Data marts indépendants (Kimball, 2013)

c. D'autres variations d'architectures

D'autres architectures peuvent exister, caractérisées en particulier par l'absence de l'un des composants du système décisionnel (Vaisman, 2014). Le serveur OLAP, par exemple, peut être absent et les requêtes utilisateurs accèdent au data warehouse directement. Dans d'autres, le data warehouse peut être également absent, et les utilisateurs accèdent aux sources de données opérationnelles ou à des vues matérialisées créées pour améliorer l'accès, qu'on appelle un data warehouse virtuel. Dans d'autres, la zone de chargement est absente ou fait partie du DW. Cependant, bien quelles soient présentes dans le monde réel, ces architectures présentent plusieurs inconvénients, notamment les lenteurs de temps de réponses, la difficulté de maintenance, l'incohérence des données, l'absence de sécurité et l'impact sur les performances des systèmes opérationnels. Ces architectures ne peuvent pas être considérées comme des architectures BI du fait qu'elles ne répondent pas aux exigences d'un système décisionnel.

1.3. Modélisation multidimensionnelle

Pour modéliser un DW, il existe deux types de modélisation pricipaux:

Schéma en étoile : appelé aussi schéma multidimensionnel (Inmon, 2005). Il est composé d'une table de fait et d'un ensemble de tables de dimension, correspondant chacune à une dimension. Dans ce schéma, les dimensions sont généralement dénormalisées, pouvant contenir des données redondantes, en particulier dans le cas de hiérarchies (Vaisman, 2014). Par exemple, le nom de la catégorie de produit et son code qui se répétent dans toutes les lignes de la table « Produits ». Par contre, la table de fait est généralement normalisée, sa clé est la combinaison des clés primaires de ses dimensions (Vaisman, 2014). Dans le cas d'une granularité fine, la clé de la table de fait peut être une colonne dédiée.

Schéma en flocon de neige : appelé aussi schéma relationnel. Dans ce schéma, les tables de dimension sont plutôt normalisées. Une table de dimension peut contenir plusieurs tables reliées entre elles par des contraintes d'intégrité référentielle. Ces tables de dimension sont reliées à la table de fait comme dans le schéma en étoile (Vaisman, 2014).

Enfin, on appelle **schéma en constellation**, un schéma contenant plusieurs tables de fait qui partagent des dimensions. Il peut contenir à la fois un schéma normalisé et un schéma dénormalisé (Vaisman, 2014).

Chacun de ces schémas posséde des avantages et des inconvénients. Le schéma en étoile, grâce à sa dénormalisation, offre de bonnes performances. Toutefois, il consomme de l'espace disque et il est difficile à maintenir en terme de conformité. Le modèle en flocon de neige permet, quant à lui, de miniser l'espace disque. Cependant, il pénalise les performances des requêtes SQL à cause de la multitude des jointures, en particulier, dans le cas de requêtes nécessitant de parcourir des hiérarchies. Un autre avantage de ce schéma, est la réutilisabilté des tables de dimensions dans plusieurs data marts. Par exemple, en créant une table catégorie de produits séparée de la table produits, on pourrait l'utiliser dans d'autres DMs consolidés par catégorie de produits et donc permettre aussi de relier plusieurs DMs.

Le choix d'un schéma reste alors très subjectif. Pour ceux qui défendent la normalisation, le schéma en flocon de niège est le plus adéquoit vu sa fléxibilité et donc son adéquation aux projets à large étendue, entreprise par exemple, contrairement au schéma multidimensionel où il est question de département ou de sous département (Inomn, 2005). Ainsi, le modèle relationnel est le mieux adapté à la conception des DWs alors que le modèle en étoile l'est pour les data

marts (Inomn, 2005). Tandis que ceux qui défondent la dénormalisation, préconisent le modèle en étoile, vu sa performance et sa simplicité, représentant deux éléments primordiaux dans la mise en place d'un data warehouse (Kimball, 2013), contrairement à la complexité du modèle relationnel, vis-à-vis des utilisateurs mais aussi des outils (analyseur des requêtes, création des indexes). Ajoutant enfin, le volume négligeable à optimiser avec le modèle en flocon de neige qui ne dépasse pas le 1% de la taille globale du DM (Kimball, 2013).

1.4. Data warehouse vs cube OLAP

Un data warehouse est un modèle dimensionnel implémenté dans une base de données relationnelle, alors qu'un cube OLAP est implémenté dans une base de données multidimensionnelle (OLAP) (Kimball, 2013), tel qu'il est illustré par la Figure 1.5. Les deux modèles partagent le même modèle logique, mais diffèrent au niveau de l'implémentation physique. En effet, lorsque les données sont chargées dans un cube OLAP, alors elles sont stockées et indexées sous un format et via des techniques propres aux systèmes OLAP. Plus précisément, les agrégations sont pré-calculées, stockées et gérées par le serveur OLAP dans des tables ou tableaux. Par conséquent, ils fournissent de meilleures performances, grâce aux agrégations pré-calculées, stratégies d'indexation et d'autres méthodes d'optimisation. Ainsi, les utilisateurs peuvent effectuer leurs analyses en parcourant les cubes vers le haut ou vers le bas (drill down, roll up), en ajoutant ou en supprimant des attributs sans écrire ou émettre de nouvelles requêtes. Outre cela, les cubes OLAP fournissent des opérations d'analyse beaucoup plus avancées et robustes que celles disponibles avec le langage SQL (Kimball, 2013). Les cubes OLAP permettent, également, une meilleure gestion de la sécurité, pouvant aller jusqu'aux données atomiques. Cependant, ils doivent être, à chaque fois, retraités pour être mis à jour. Enfin, contrairement aux bases de données relationnelles, l'implémentation et le déploiement des cubes OLAP sont très liés à l'outil ou la plateforme choisie (différents raisonnement, différentes capacités etc.). D'où la difficulté de transporter des cubes d'une plateforme vers une autre contrairement aux bases de données relationnelles.

On distingue deux modes de stockage principaux pour les cubes OLAP (Gorla, 2003):

MOLAP (Multidimensional OLAP): Dans ce mode, les données et agrégations sont pré-calculées et stockées sur le serveur OLAP sous forme de tableaux (Cheung et al, 2003) permettant ainsi de garantir les meilleures performances. La seule contrainte dans ce mode est, néanmoins, celle de l'espace de stockage requis.

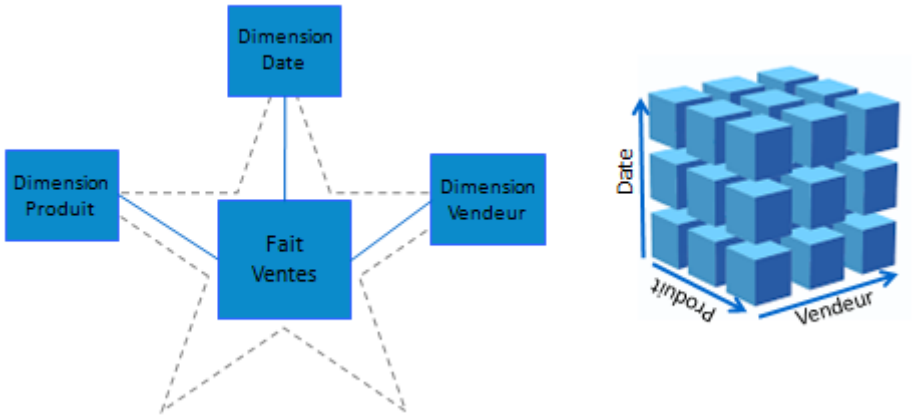


Figure 1.5 Schéma en étoile vs cube OLAP (Kimball, 2013)

ROLAP (Relational OLAP): dans ce mode, les données sont stockées sur le serveur relationnel ainsi que les agrégations. Ces dernières sont pré-calculées et stockées séparément dans des tables relationnelles appelées tables d'agrégation. Le mode ROLAP ne requiert alors pas d'énorme espace de stockage. Son inconvénient est, cependant, la lenteur de traitement des requêtes multidimensionnelles, du fait qu'elles doivent être traduites en requêtes SQL, généralement très complexes (Vaisman, 2014).

Enfin, il existe d'autres variations de ces deux modèles de base, MOLAP et ROLAP, notamment le mode **HOLAP** (Hybrid OLAP). Ce dernier est un mode hybride permettant de profiter de l'avantage de stockage du ROLAP en stockant les données détaillées sur le serveur relationnel, et des performances de traitement fournies par le MOLAP en stockant les agrégations sur le serveur OLAP.

1.5. Outils OLAP

Dans un projet décisionnel, le choix des outils est un élément primordial et décisif. En effet, contrairement aux éditeurs des bases de données relationnelles, les structures OLAP diffèrent d'un outil à l'autre. Ces derniers ont chacun son environnement personnalisé, ses capacités et ses limitations, mais surtout son propre jargon. Entre outils ROLAP, manipulant des tables et des jointures avec du langage SQL, comme Mondrian, à des outils complètement multidimensionnels, comme SSAS et autres, qui manipulent des données et agrégations via des langages dédiés comme le MDX (MultiDimensional eXpressions). La portabilité des applications BI entre les outils OLAP, s'avère donc quasiment impossible (Kimball, 2013). De ce fait, les compétences des développeurs BI restent très liées aussi à l'outil utilisé.

Nous avons mené dans ce sens une étude des outils OLAP les plus utilisés. Le résultat de cette étude est présenté dans le tableau 1 ci-dessous:

Tableau 1.1 Comparaison de quelques outils OLAP

Editeur	Outil OLAP	Support de la relation plusieurs-à- plusieurs	Mode de stockage supporté	Support du partitionnement	Format des fichiers OLAP
Microsoft	Microsoft SQL Server Analysis Services	Oui	MOLAP ROLAP HOLAP	Oui	XMLA
Pentaho	Mondrian OLAP Server	Non	ROLAP	Oui	XMLA
SAS Institute	SAS OLAP Server	Non	MOLAP ROLAP HOLAP	Oui	PROC OLAP
SAP	NetWeaver Business Warehouse	Non	MOLAP ROLAP	Oui	XML
IBM	Cognos TM1	Non	MOLAP ROLAP HOLAP	Oui	XMI
icCube	icCube	Oui	MOLAP	Oui	XMLA
ORACLE	Oracle Database OLAP Option	Oui	MOLAP ROLAP HOLAP	Oui	XML

1.6. Les Métadonnées

Les métadonnées sont des informations décrivant la structure et la signification des données et des applications, ainsi que les processus qui les manipulent (MDC-OIM, 1999). Ces métadonnées jouent un rôle primordial dans la réussite de la gouvernance des données (Informatica white paper, 2013). Cette dernière est la discipline de contrôle qualité pour l'évaluation, la gestion, l'utilisation, l'amélioration, la surveillance, la maintenance et la protection des informations organisationnelles (Johnson, 2011). Khatri et al distinguent cinq domaines de décision dans la gouvernance des données (Khatri et al, 2010), à savoir: les principes des données, la qualité des données, le cycle de vie des données, l'accès aux données et, bien sûr, les métadonnées. En effet, les métadonnées sont indispensables dans plusieurs domaines des systèmes décisionnels. Tout d'abord, elles sont utilisées comme langage commun entre développeurs BI et utilisateurs finaux en mappant le vocabulaire métier à celui technique,

qu'on appelle un glossaire. Les métadonnées peuvent être utilisées également dans la documentation du système mais aussi dans la mise en place des reporting et tableaux de bord nécessitant l'utilisation du langage métier.

Les métadonnées dans les systèmes décisionnels sont également utilisées pour tracer la lignée de données de bout en bout, ou ce qu'on appelle la carte des données, permettant de suivre les données depuis leur source jusqu'à leur destination finale. Ceci permet d'aider les développeurs dans différents usages, notamment, l'estimation de l'impact de changement, la traçabilité des changements et l'identification des origines d'incohérences. D'où l'importance de la gestion des métadonnées dans les DSS. Pour ce faire, des outils dédiés peuvent être utilisés (Marco et al, 2004) ou simplement des fichiers ou tableaux personnalisés. Ces deux moyens nécessitent, toutefois, une intervention humaine, en particulier pour définir le mapping avec les sources de données et fournir les descriptions métier, ainsi que le traçage de la lignée des données. Malheureusement, cette tâche est souvent négligée par les développeurs BI, ce qui impacte l'évolutivité du DSS et augmente son coût de maintenance.

1.7. Cycle de développement des systèmes décisionnels

Le cycle de conception d'un système décisionnel comprend les étapes suivantes (Figure 1.6) :

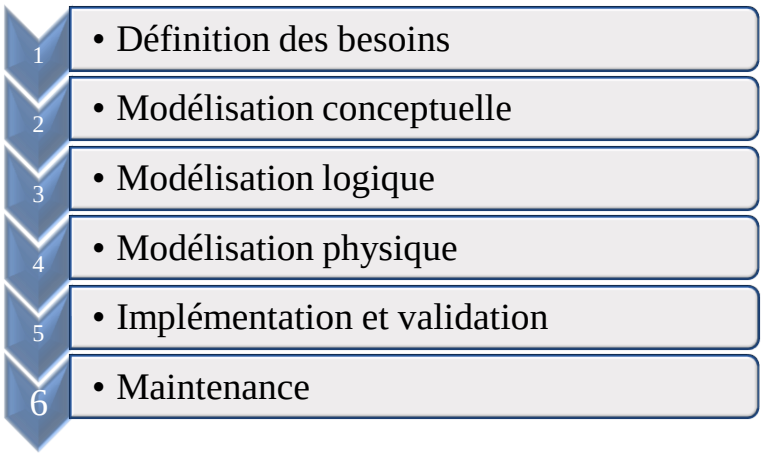


Figure 1.6 Cycle de développement d'un DSS

Définition des besoins utilisateurs: elle consiste à collecter les informations relatives aux besoins des décideurs. Contrairement au développement logiciel, où l'utilisateur définit précisément les fonctionnalités du nouveau système, dans le cas d'un DSS, les besoins sont exprimés sous forme de buts stratégiques que les décideurs souhaitent réalisés. Le concepteur BI doit, dans ce cas, transformer ce besoin en spécifications précises. On parle alors d'ingénierie des

besoins (Requirements Engineering) (Prakash et al, 2008). La définition des besoins utilisateurs peut être effectuée en deux étapes: d'abord, la définition des exigences de haut niveau (Early Requirements ER) qui est une vision perspicace des buts stratégiques définis par le top management. Par la suite, analyser et enrichir ces exigences afin d'obtenir une vue détaillée, appelée exigences de bas niveau (Late Requirements LR). Cette phase peut inclure aussi l'identification des sources de données opérationnelles ainsi que la réconciliation de ces dernières avec les besoins exprimés. Par ailleurs, et pour faciliter la compréhension du processus métier par les développeurs, l'utilisation d'ontologies décrivant les sources de données et le processus métier est préconisée (Pradillo, 2011).

Pour réaliser ce modèle, plusieurs techniques existent, à savoir: les techniques informelles comme les questionnaires et les cahiers de charges et les techniques formelles ou semi formelles comme Merise via ses modèles conceptuels MCD et MCT et l'UML via son digramme de cas d'utilisation (Luján-Mora et al, 2006) et digramme d'activité (Zepeda et al, 2008), ainsi que le Framework i star (i*) largement utilisé dans la littérature (Mazon et al, 2007). Des représentations graphiques peuvent également être utilisées comme le graphe GDI (Goal-Decision-Information) (Prakash et al, 2008) et le graphe GRT (Goal Refinement Tree) (Zepeda et al, 2008).

Modèle conceptuel : Ce modèle prend en entrée les besoins des utilisateurs collectés précédemment. Il a pour but de fournir une représentation du nouveau DSS sans prendre en considération les contraintes d'implémentation (Vaisman, 2014). A ce niveau, le degré de granularité du data warehouse doit être choisi.

En fait, la granularité du DW est le niveau de détail d'une unité de données dans le DW (Inmon, 2005). Plus précisément, la granularité est ce que représente une ligne de la table de fait. La définition de la granularité est l'un des aspects les plus importants dans la conception d'un DW et dont l'impact se répercute sur toute son architecture (Inmon, 2005; Kimball, 2013). Le choix de la granularité impacte également le volume de données et le type de requêtes pouvant être répondues par le DW. Ainsi, il est toujours recommandé d'opter pour une granularité très fine (Inmon, 2005; Kimball, 2013; Ponniah, 2004) c.à.d. pour le plus bas niveau de détail fourni par les processus métier ou par les sources de données opérationnelles, afin de permettre une extensibilité facile du DW. Par exemple, supposons que le besoin actuel des utilisateurs est d'analyser les ventes par mois, la conception du DW ne doit pas se restreindre à ce besoin, mais doit plutôt tenir compte du niveau de détail disponible dans les sources de données, niveau date par exemple. Ainsi, le DW pourra répondre à tout nouveau besoin d'analyse par date.

Enfin, pour réaliser ce modèle conceptuel, plusieurs techniques de modélisation peuvent être utilisées, dont les plus répandues sont le modèle entité/association et le diagramme de classe UML. Dans la littérature, plusieurs travaux ont également utilisé le modèle conceptuel graphique DFM (Dimensional Fact Model) (Golfarelli, 1998) composé d'un fait (ensemble de mesures) et d'arbres d'attribut, représentant chacun une dimension ainsi que ses attributs et hiérarchies.

Modèle logique : il est la traduction directe, voir automatique, du modèle conceptuel en un modèle spécifique à un type de plateforme pouvant être implémenté. Il s'agit du modèle relationnel commun à la plupart des SGBDs ou du modèle multidimensionnel commun aux plateformes OLAP. Pour représenter le modèle relationnel, le modèle E/A est le plus souvent utilisé.

Pour le modèle OLAP, il existe deux standards de modélisation, le CWM (Common Warehouse Model) et l'OIM (Open Information Model) qui proposent chacun un métamodèle pour représenter ce modèle OLAP (voir chapitre 2).

Modèle physique : c'est un modèle très proche du modèle logique mais qu'on enrichie avec plus de contraintes techniques, en particulier celles relatives à la plateforme ou l'éditeur choisi. Très peu d'approches utilisent ce modèle, la plupart se contentent du modèle logique pour passer à l'implémentation.

Cette conception à quatre niveaux permet de séparer la phase de modélisation de la phase de développement et de garantir ainsi que les modifications apportées aux niveaux inférieurs n'affecteront pas les niveaux supérieurs (Vaisman, 2014).

Implémentation et validation : désigne la création effective du système et, par la suite, sa validation incluant l'assurance de la qualité des données, le traitement des opérations et les tests de performance (Kimball, 2013).

Maintenance : se fait tout au long du cycle de vie du système et à fur et à mesure de son évolution. Les tâches de maintenance incluent la gestion du chargement du DW et cubes OLAP, la gestion de l'espace de stockage, l'optimisation, la réplication etc.

1.8. Démarches de mise en place des systèmes décisionnels

Mettre en place un système décisionnel est un processus qui comprend plusieurs étapes, à savoir la modélisation, la création de l'entrepôt de données, l'identification des sources de données opérationnelles, la création des outils de chargement et de transformation des données, la création des cubes OLAP et enfin l'implémentation de reporting et de tableaux de bord à mettre à disposition de l'utilisateur final. Pour réaliser ce système, deux types principaux de démarches existent :

Approche descendante (Top-down) : elle consiste en l'adoption d'une vue d'entreprise du système décisionnel dans le cadre d'un seul et unique projet, permettant ainsi une gestion centralisée des données et des règles du projet. Elle peut, par contre, générer un coût et un délai de mise en place très importants, en plus de l'augmentation des risques d'échec (Ponniah, 2004), en particulier dans le cas de développeurs inexpérimentés (Vaisman, 2014). Le système est ainsi modélisé à partir des sources de données opérationnelles, ou en rassemblant les besoins de tous les utilisateurs.

Approche ascendante (Bottom-up) : Dans ce type d'approche, le système décisionnel est mis en place progressivement, suivant les besoins des utilisateurs, permettant ainsi une implémentation facile et rapide de segments gérables du projet et un retour sur investissement favorable et rapide, mais également, la réduction des risques d'échec du projet (Ponniah, 2004). Toutefois, cette approche peut présenter aussi des inconvénients, en particulier dans le cas de data marts indépendants, parmi lesquels la redondance des données et leur incohérence.

Approche mixte : Etant donné les avantages et inconvénients de chacune des deux approches ascendante et descendante, l'approche mixte ou hybride serait la plus pratique mais aussi la plus recommandée (Vaisman, 2014). Elle consiste en l'adoption d'une vue globale ou vue d'entreprise du projet qui comprend les différents processus métier, les métriques, le jargon commun etc. Ensuite, mettre en place le système décisionnel progressivement, tout en s'alignant toujours avec cette vue globale, afin de créer des data marts cohérents et pouvant être utilisés par différents utilisateurs.

En dessous de ces trois grandes catégories qui définissent le processus global de mise en place des systèmes décisionnels, nous distinguons trois types d'approches qui définissent cette fois-ci l'orientation du projet :

Approche dirigée par les données (data driven approach) : elle consiste à analyser les sources de données opérationnelles dans le but d'identifier les données disponibles et susceptibles d'intéresser les décideurs (Saroop et al, 2011). Le coût et le délai de mise en place est donc réduit vu la simplification du processus de chargement des données (ETL). Cependant, elle nécessite une connaissance approfondie des sources de données et des processus métier par les développeurs.

Approche dirigée par les besoins des utilisateurs (requirement driven approach) : se base sur l'implication des utilisateurs à définir leurs besoins, que les développeurs doivent combiner par la suite, dans un schéma global. La réconciliation avec les sources de données est faite à postériori. Bien que ce type d'approche soit largement apprécié par les utilisateurs finaux grâce à

leur implication, il présente aussi des inconvénients, notamment, la divergence des besoins utilisateurs ainsi que la difficulté d'adapter ces besoins aux schémas des sources de données.

Approche dirigée par les buts (goal driven approach) : se focalise sur les stratégies business définies par le top management. Les différentes visions sont donc analysées et fusionnées pour définir le schéma global du système décisionnel, caractérisé dans ce cas par l'intégration des indicateurs clés de performance (Letrache et al, 2016), servant à suivre l'évolution des buts stratégiques.

1.9. Etat de l'art sur les démarches d'élaboration des systèmes décisionnels

Comme vu précédemment, plusieurs types de démarches de mise en place des systèmes décisionnels existent et qui diffèrent aussi bien dans l'orientation que dans la portée du projet. Toutefois, aucun standard n'a été établi. Ainsi, plusieurs travaux dans la littérature ont tenté de proposer des techniques et des méthodologies qui couvrent l'élaboration d'un ou plusieurs composants du système décisionnel, que nous pouvons classer en deux catégories : manuelle et automatique.

1.9.1. Démarches manuelles

La mise en place manuelle des systèmes décisionnels est la démarche classique mais aussi la plus répondue, en particulier dans le domaine professionnel. Elle consiste en le passage manuel d'une phase du cycle de vie du projet vers l'autre selon le type d'approches adopté. Etant donné l'absence d'un standard de développement des DSS, plusieurs auteurs ont tenté de dresser des directives pour guider la conception et la mise en place d'un DSS:

Ralph Kimball : un des grands auteurs et experts des systèmes décisionnels. Ses livres constituent un référentiel pratique pour les développeurs et administrateurs BI. Il est considéré comme étant le propriétaire de l'approche multidimensionnelle (Inmon, 2005). Il est aussi l'un des premiers auteurs à préconiser l'approche ascendante guidée par les exigences métier dont il définit les grandes lignes comme suit (Kimball, 2011, 2013) :

- Planifier et définir les exigences au niveau global de l'entreprise
- Créer l'architecture d'un entrepôt de données complet
- Conformiser et standardiser le contenu des données
- Implémenter l'entrepôt de données en une série de super-data marts.

Il définit aussi quatre étapes pour la modélisation dimensionnelle: (1) d'abord, identifier le processus ou l'activité métier à analyser. (2) définir ensuite la granularité des données. Pour cela,

l'auteur recommande le choix d'une granularité très fine (atomique) dans le but de répondre aux besoins imprédictibles des utilisateurs. (3) l'étape suivante est le choix des dimensions ou axes d'analyse qui doivent être en cohérence avec la granularité choisie précédemment. (4) la dernière étape est l'identification des faits ou mesures, c.à.d. des données, généralement numériques, à analyser.

William H. Inmon : considéré comme le père du concept de Data warehousing. Son approche complètement différente de celle de Kimball a suscité plusieurs débats. En effet, les deux auteurs diffèrent aussi bien dans l'architecture que dans la démarche de mise en place. En fait, l'approche d'Inmon est une approche descendante qui traite le DW comme un projet d'entreprise centralisé. Son approche est guidée par les modèles des sources de données et suit un cycle de vie de développement complètement opposé au cycle de développement traditionnel, qu'il appelle CLDS (l'inverse de SLDC) et dont il définit les principales phases comme suit (Inmon, 2005) :

- Implémenter l'entrepôt de données à partir des modèles des sources de données opérationnelles
- Intégrer les données
- Tester ces données pour retrouver les tendances si elles existent
- Programmer ensuite, à la base de ces données
- Analyser les résultats
- Comprendre finalement le besoin utilisateur

Recommencer ensuite ce même processus pour tout autre ensemble de données, ce que l'auteur appelle une méthode de développement « spirale ».

Golfarelli et al : ont proposé une approche dirigée par les données pour le développement des DWs, composée de six phases. La première phase est l'analyse des systèmes opérationnels en collaboration avec leurs responsables afin de produire les schémas conceptuels ou logiques des bases de données opérationnelles. La deuxième phase consiste à identifier, en collaboration avec les utilisateurs, les faits à partir des schémas produits dans la première phase. Ensuite, établir une définition préliminaire des requêtes utilisateurs dans un langage semi-naturel. La troisième phase consiste en la modélisation conceptuelle du DW en se basant sur les schémas déduits des sources de données, ainsi que les faits et les requêtes utilisateurs définies auparavant. Le modèle résultant est un schéma DFM (Dimensional Fact Model) (Golfarelli et al, 1998). La quatrième étape concerne le raffinement des requêtes utilisateurs en rajoutant les détails requis selon le schéma conceptuel obtenu, ainsi que le calcul du volume de données du futur DW. A partir de ces éléments, le modèle logique (relationnel ou multidimensionnel) est déduit. Cette cinquième

phase, inclut également la définition des vues matérialisées et de la stratégie de partitionnement dans le cadre de l'optimisation du futur DW. Enfin, le modèle physique est généré. Ce dernier traite en particulier la sélection des indexes dans le cas de DW relationnel.

Prakash et al : dans le cadre d'une approche dirigée par les buts, les auteurs proposent d'utiliser le modèle GDI (Goal-Decision-information) pour la conception des systèmes décisionnels (Prakash et al, 2008, 2018). Dans ce modèle, les buts sont des composants passifs qui requièrent des décisions (composants actifs) pour être réalisés. De même, les décisions, qui représentent un ensemble d'actions, nécessitent des informations pour leur accomplissement. Ces dernières sont déduites des besoins utilisateurs exprimés en langage SQL simplifié, que les auteurs nomment SSQL (Specification SQL). La sortie de la phase des exigences est le modèle GDI, alors que la définition des faits, mesures, dimensions est faite dans le modèle conceptuel. Toutefois, le modèle ignore les acteurs intervenants, positivement ou négativement, dans chaque décision ou but.

Giorgini et al : proposent une approche dirigée par les buts, utilisant la méthode Tropos basée sur le Framework i* (Giorgini et al, 2008) . Le modèle comprend les acteurs responsables de l'achèvement des buts, en plus des dépendances entre ces acteurs. Ainsi, pour analyser les dépendances entre les acteurs, un diagramme d'acteur est utilisé. Par la suite, Les buts relatifs à chaque acteur, sont détaillés et décomposés en sous buts reliés par les opérateurs « ET » ou « OU » dans un diagramme rationnel. Ce dernier est ensuite étendu pour inclure les faits, mesures et dimensions associés à chaque but. Enfin, le modèle rationnel est réconcilié avec les modèles des bases de données opérationnelles pour générer un modèle multidimensionnel représenté avec le schéma DFM (Golfarrelli et al, 1998).

1.9.2. Démarches semi-automatiques

Afin de réduire le coût et délai de mise en place et de maintenance des systèmes décisionnels plusieurs travaux dans la littérature ont proposé d'automatiser ce processus ou l'une de ses étapes. Nous citons ci-dessous quelques une de ces approches :

Romero et al : proposent un Framework dirigé par les données pour la conception semi-automatique du modèle multidimensionnel des entrepôts de données (Romero et al, 2007, 2011). L'approche proposée consiste à analyser en premier lieu, les sources de données à partir desquelles les éventuels faits, dimensions et hiérarchies sont déduits automatiquement. Ces derniers sont décrits avec une ontologie en langage OWL (Ontology Web Language) dont le domaine est les sources de données. Un premier filtrage des schémas obtenus est, par la suite, réalisé automatiquement via des algorithmes proposés par les auteurs (Romero et al, 2010).

Enfin, à partir des schémas restants, l'utilisateur final choisi le schéma qui s'adapte le mieux à ses besoins.

Jovanovic et al : proposent, comme continuité de l'approche proposée par Roméro et al, une approche itérative, semi-automatique pour la conception du modèle multidimensionnel (Jovanovic et al, 2014). L'approche proposée consiste à utiliser comme entrée une ontologie pour décrire les sources de données en langage OWL (Ontology Web Language). Par la suite, et via un traitement itératif, intégrer les besoins utilisateurs exprimés avec une grammaire prédéfinie. Les utilisateurs interviennent, donc, dans chaque étape pour valider les modèles multidimensionnels obtenus. L'approche permet aussi, lors de ce traitement itératif, d'unifier plusieurs schémas en un seul contrairement à l'approche initiale. Les auteurs fournissent l'ensemble des algorithmes nécessaires au traitement d'unification des modèles (Ex. opération d'ajout de mesures ou d'attribut de dimension, fusion de tables de fait etc.). La solution proposée intègre aussi la notion de coût qualitatif de la solution que l'utilisateur peut utiliser pour choisir le meilleur modèle. La solution proposée présente, cependant, l'inconvénient de faire intervenir l'utilisateur final dans toutes les phases de conception, ce qui peut être hors son domaine de compétence.

Pinet et al : proposent dans leur approche semi-automatique dirigée par les données (Pinet et al, 2009), un nouveau modèle multidimensionnel UML qui permet de modéliser, à la fois, fait et dimension avec le même concept. L'approche proposée débute par la génération automatique d'un premier modèle multidimensionnel à partir des sources de données, en utilisant ce que les auteurs appellent un «Helping Development System ». Après plusieurs étapes de vérification, de validation et d'intégration des contraintes OCL, le modèle relationnel ainsi que le code SQL correspondant sont générés via l'outil OCLtoSQL.

Song et al : proposent un outil basé sur leur approche automatique dirigée par les données pour la conception des DWs (Song et al, 2008). L'approche proposée consiste à générer le schéma en étoile du DW à partir des schémas E/A des sources de données, comme suit : à partir de la représentation graphique des sources de données, extraire la liste des entités, associations et attributs en utilisant le parseur JAXP (Java API for XML processing). Par la ensuite, déduire via un algorithme, les tables de faits et de dimensions en se basant sur leurs cardinalités (Song et al, 2007).

Prat et al : proposent une approche semi-automatique dirigée par les besoins utilisateurs et qui a pour objectif de générer les modèles conceptuel, logique et physique du DW (Prat et al, 2006). Ainsi, la première phase de l'approche proposée consiste à collecter les besoins utilisateurs via des interviews et des prototypes de rapports. Ensuite, modéliser ces besoins avec un diagramme

de classe UML. La deuxième phase est de générer un modèle logique représenté par un modèle multidimensionnel à partir du modèle conceptuel, enrichi avec des concepts multidimensionnels. Enfin, un modèle physique relatif à la plateforme choisie est généré. Dans chaque phase, les auteurs proposent un métamodèle auquel doit se conformer le modèle résultant ainsi qu'un ensemble de transformations exprimées en langage OCL et permettant le passage automatique d'un niveau vers l'autre.

1.10. Conclusion

Dans ce chapitre, nous avons présenté le système décisionnel et ses différents composants. Nous avons présenté aussi les différentes architectures possibles et les contraintes et alternatives d'élaboration de ce système.

En effet, il existe plusieurs approches de mise en place d'un DSS, qui diffèrent aussi bien dans la portée du projet que dans l'architecture, l'orientation et le niveau d'automatisation.

Nous avons présenté alors la première classification de ces approches, à savoir les approches ascendantes, descendantes et mixtes, parmi lesquelles, l'approche mixte serait la plus recommandée (Kimball, 2013 ; Vaisman, 2014).

Au sein de ces trois grandes catégories, nous avons présenté trois autres classes, à savoir les approches dirigées par les données, dirigées par les utilisateurs et celles dirigées par les buts. En dressant les avantages et inconvénients de chacune, l'approche dirigée par les buts s'avère la plus adaptée aux projets décisionnels.

Une dernière distinction concerne cette fois-ci le niveau d'automatisation de ces approches. On distingue alors entre approches manuelles et approches automatiques.

En fait, les approches automatiques, ou semi automatiques, permettent un gain important en termes de coût et de délai de développement, vu l'évolutivité des projets décisionnels et la répétitivité des tâches de mise en place de nouveaux data marts. Ainsi, plusieurs travaux dans la littérature ont proposé des algorithmes et des outils permettant d'automatiser le processus de développement d'un DW en utilisant des langages comme l'OWL-DL ou l'OCL. Beaucoup se sont basés sur les ontologies des sources de données pour passer au modèle conceptuel ou logique (Romero et al, 2011; Jeovanovic et al, 2014; Pinat et al, 2009). Toutefois, ces approches peuvent entrainer des conceptions qui ne sont pas optimisées ou qui sont redondantes (le cas, par exemple, de plusieurs tables de fait pouvant être regroupées dans une seule). De plus, ces approches utilisent des langages limités et non dédiés, qui requièrent donc des développements compliqués.

Pour pallier à cela, une des solutions prometteuses est l'architecture dirigée par les modèles ou MDA (Model Driven Architecture) grâce à ses nombreux avantages. Cette dernière, basée sur le concept de méta-modélisation, permet la réutilisabilité et la pérennité des modèles, en plus de l'automatisation du processus de développement, via des langages de transformations dédiés et performants.

Tenant compte des avantages et inconvénients de chaque type d'approches, notre proposition est une approche mixte, dirigée par les buts et automatisée grâce à l'adoption d'une architecture MDA que nous présentons en détail dans le chapitre qui suit.

Chapitre 2

La MDA dans le développement des projets décisionnels

Résumé : l'ingénierie dirigée par les modèles est une discipline qui suggère l'utilisation systématique des modèles dans le développement logiciel. Une des initiatives les plus intéressantes dans ce domaine est la MDA (Model Driven Architecture) développée sous l'égide de l'OMG (Object Management Group). La MDA a été largement utilisée dans la littérature dans le cadre de l'élaboration des systèmes décisionnels. Ce chapitre décrit les principaux concepts de cette architecture ainsi que son application dans le domaine de l'informatique décisionnelle.

2.1. Définition

L'architecture dirigée par les modèles (ou MDA pour Model Driven Architecture) a été proposée par l'OMG (Object Management Group) en 2000 comme solution au problème d'interopérabilité et de portabilité des applications (OMG-MDA, 2001 ; Bézivin et al, 2002). En effet, la MDA offre une approche ouverte et neutre par rapport aux éditeurs, afin de faire face aux changements business et technologiques. L'approche MDA fait partie des processus de l'ingénierie dirigée par les modèles (ou MDE pour Model Driven Engineering) (Bézivin et al, 2004), qui se réfère à l'utilisation systématique des modèles comme des éléments centraux, tout au long du cycle de vie du logiciel. La MDA permet ainsi de séparer les spécifications d'une fonctionnalité du système des spécifications relatives à l'implémentation de cette fonctionnalité dans une plateforme spécifique. Ainsi, la MDA, grâce aux standards sur lesquels elle repose, notamment l'UML, MOF et CWM (voir Figure 2.1), permet de réaliser un même modèle sur différentes plateformes via un mapping défini. Elle permet également l'interopérabilité entre les applications en reliant explicitement leurs modèles.

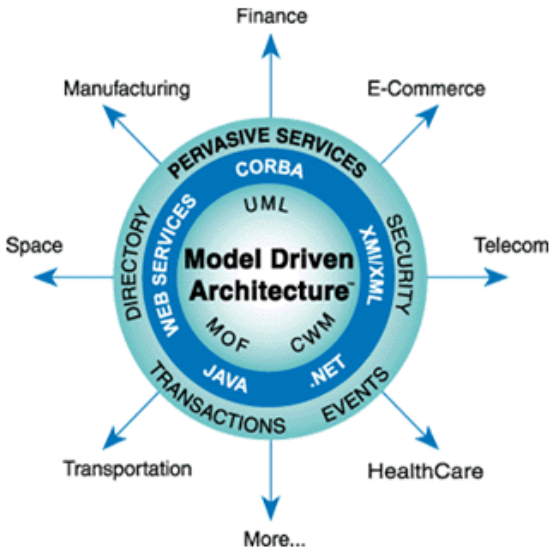


Figure 2.1 L'architecture MDA (OMG-MDA, 2001)

La MDA permet d'extraire de la valeur depuis les modèles et l'architecture utilisés, tout au long du cycle de développement d'une application, depuis la définition des besoins, puis la modélisation, jusqu'à son implémentation. Elle permet ainsi de faire face à la complexité et l'interdépendance des systèmes complexes (OMG-MDA, 2014).

L'architecture MDA, comme le reste des processus MDE, se base sur les concepts de base suivants :

Abstraction: elle traite l'habilité de comprendre un système de manière plus générale. Plus concrètement, l'abstraction permet d'introduire une vision de haut niveau au détriment des détails. Ainsi, un modèle est dit plus abstrait s'il englobe un ensemble plus large de systèmes, et moins abstrait s'il est plutôt spécifique à un système unique ou à un ensemble restreint de systèmes (OMG-MDA, 2014).

Perspective : c'est une abstraction technique permettant de se focaliser sur un aspect particulier du système tout en éliminant les détails non importants. Une perspective peut être représentée par un ou plusieurs modèles (Cephas, 2006).

Modèle : dans le contexte MDE, un modèle est une description d'un système (ou partie du système) écrite dans un langage bien défini. Un langage bien défini est un langage ayant une forme (syntaxe) et une sémantique bien définies, qui convient à l'interprétation automatique par un ordinateur (Kleppe, 2003). L'OMG définit un modèle comme étant une information représentant de manière sélective quelques aspects d'un système dans un but précis. Il est relié au système via un mapping implicite ou explicite. Un modèle devrait inclure l'ensemble des informations sur un système en question en plus des règles d'intégrité qui s'appliquent à ce système et la signification des termes utilisés. Enfin, un modèle peut représenter le métier, le domaine, le logiciel, le matériel, l'environnement ou autres aspects spécifiques d'un système (OMG-MDA, 2014).

Métamodèle : c'est un modèle qui définit un langage de modélisation (voir Figure 2.2) et qui est également exprimé en utilisant un langage de modélisation (OMG-MDA, 2014). Il peut être vu comme étant la spécification de tous les modèles possibles dans ce langage de modélisation (OMG-MDA, 2010).

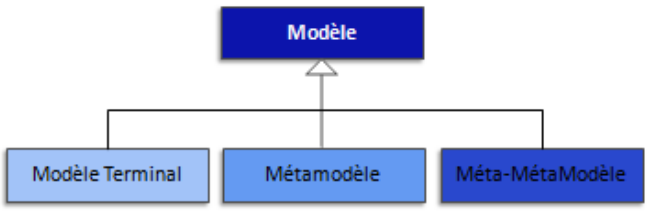


Figure 2.2 Les types de modèles de l'OMG

Pour définir ses standards, l'OMG utilise une architecture à quarte couches qu'elle nomme M0, M1, M2 et M3 (Bézivin et al, 2004) comme illustré par la Figure 2.3. Dans la couche M0, on représente le système en cours d'exécution, là où sont les données réelles du système, par exemple : le client «*Amine*» habitant à «*Casablanca*». Il existe généralement plusieurs instances

de client ayant chacune ses propres données. Dans la couche M1, on retrouve des modèles, comme un modèle UML qui représente une application. Par exemple, on retrouve dans la couche M1 le concept ou la classe *Client*, caractérisée par un nom, ville etc. La couche M1 peut être considérée comme une catégorisation ou classification des instances se trouvant dans la couche M0. En d'autres termes, tout élément de M0 est une instance d'un élément dans M1 (Kleppe, 2003). Ce dernier, doit être conforme à son métamodèle défini dans M2. Ce métamodèle peut être un standard comme le métamodèle du diagramme de classe UML ou personnalisé comme un profil UML qui s'adapte à un besoin particulier. Enfin, dans la couche M3, on retrouve le méta-métamodèle de celui utilisé dans M2, qui à son tour, peut être exprimé avec un méta-métamodèle dans la même couche. La couche M3 utilise le langage MOF (Meta Object Facility), qui est un langage abstrait et auto-défini pour la spécification, la construction et la gestion indépendante des métamodèles en termes de technologies. Il constitue la base pour définir n'importe quel langage de modélisation comme l'UML ou le MOF lui-même (OMG-MOF, 2016).

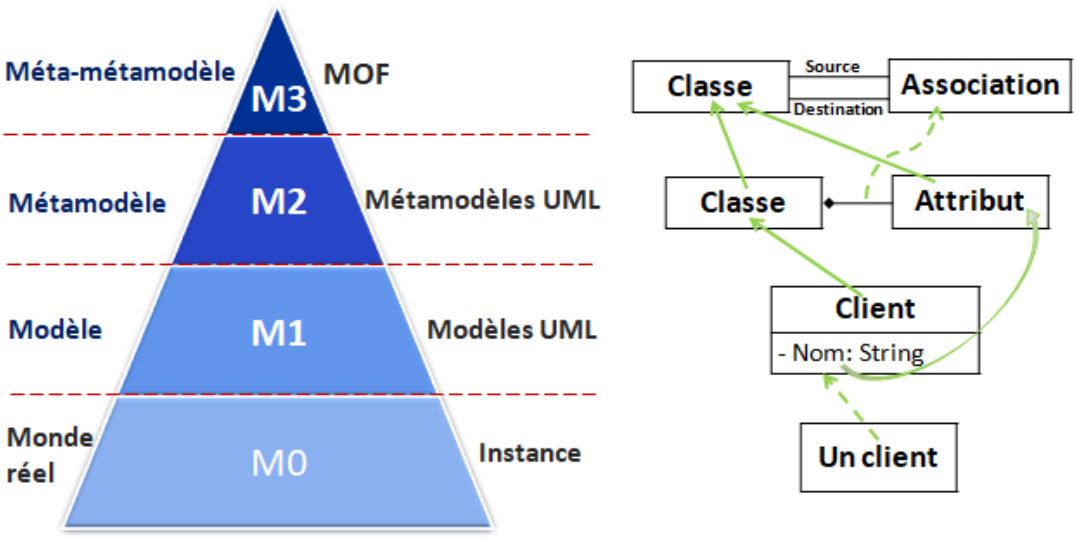


Figure 2.3 Architecture à quatre couches de l'OMG

Transformation : c'est la génération automatique d'un modèle cible à partir d'un modèle source (voir Figure 2.4), en respectant une définition de transformation. Une définition de transformation est un ensemble de règles de transformation qui décrivent la manière avec laquelle un modèle, dans le langage source, peut être transformé en un modèle dans le langage cible (Kleppe, 2003).

Figure 2.4 Transformation modèle-à-modèle

Dans le cas où le modèle source ainsi que le modèle de destination appartiennent au même niveau d'abstraction, on parle de transformation horizontale. Dans ce cas, il s'agit d'une restructuration de modèle. Par contre, si le niveau d'abstraction est différent, alors on parle de transformation verticale. On distingue aussi entre deux types de transformations qui sont les transformations modèle-à-modèle notées par M2M, où la transformation sert à générer un modèle à partir d'un autre, et les transformations modèle-à-texte notées par M2T, servant à générer du texte structuré à partir d'un modèle.

Pour implémenter ces transformations, plusieurs langages de transformation existent (Czarnecki et al, 2006), dont les plus connus et les plus utilisés sont le QVT (Query/View/Transformation) un standard proposé par l'OMG (OMG-QVT, 2016), et l'ATL (Atlas Transformation Language) proposé par ATLAS group de l'université de Nantes et la TNI-Valiosys Company, comme langage similaire au QVT (Bézivin et al, 2003) (Jouault et al, 2006). Les deux langages partagent plusieurs points en commun, que ce soit au niveau de la logique de transformation qu'au niveau syntaxique. En effet, les deux langages sont basés sur le MOF et utilisent les expressions OCL (Object Constraint Language) pour interroger les modèles. De plus, les deux langages sont de type hybride du fait qu'ils supportent les expressions déclaratives et impératives. D'autres points communs existent entre les deux modèles, notamment la modularité des règles de transformation, la création et le raffinement des modèles ainsi que la traçabilité (la source de l'élément, heure de création etc.) (Czarnecki et al, 2006).
Notant une différence entre les deux langages et qui concerne la transformation modèle-à-texte. Cette dernière est directement supportée par l'ATL (ATL, 2006), alors que le QVT nécessite l'utilisation d'un langage supplémentaire qui est le MOF2Text.

2.2. Les modèles de la MDA

La MDA définit quatre modèles de base dans son cycle en Y (voir Figure 2.5), correspondant chacun à une perspective. Ces modèles correspondent en fait à des niveaux d'abstraction, vu que dans chaque niveau, plusieurs modèles peuvent être construits, où chacun est focalisé sur un

aspect particulier (interfaces utilisateurs, architecture etc.) (Cephas, 2006). Ces modèles sont les suivants:

Computation Independent Model (CIM) appelé aussi modèle business (Kleppe, 2003), du fait qu'il emploie le même jargon que celui des experts métier. Il représente exactement ce que le système est censé accomplir, en ignorant toute spécification technique afin de rester indépendant de la manière dont le système est, ou sera implémenté. Le CIM joue un rôle important pour faire le pont entre les experts du domaine et informaticiens responsables de la mise en place du système. Dans une approche MDA, les spécifications établies dans le CIM, doivent être traçables dans le PIM ainsi que dans le PSM qui l'implémente et vice-versa (Cephas, 2006).

Platform Independent Model (PIM) il fournit les spécifications formelles de la structure et le fonctionnement du système en ignorant tout détail technique relatif à l'implémentation finale (Miller et al, 2001). Il représente un niveau élevé d'abstraction (Kleppe, 2003) et un degré suffisant d'indépendance, de telle manière à permettre sa traduction en une, ou plusieurs, plateformes cibles.

Platform Specific Model (PSM) représente une traduction du PIM dans une plateforme spécifique. Il combine les spécifications du PIM avec les détails requis pour stipuler comment le système sera implémenté dans une plateforme particulière.

Platform Description Model (PDM) contient les informations techniques requises pour dériver un PSM depuis un PIM. Il décrit l'architecture technique et les capacités d'une plateforme.

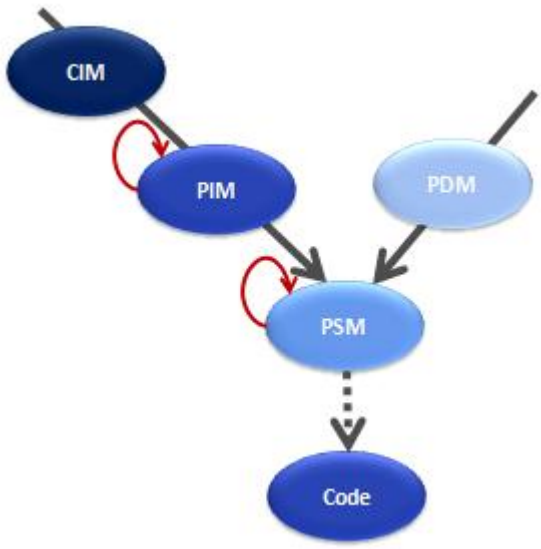


Figure 2.5 Cycle en Y de la MDA

Code source : la dernière étape dans une approche MDA est la génération du code d'implémentation du système dans une plateforme spécifique. Il peut s'agir également de la génération d'un fichier de configuration, fichier de déploiement etc. Enfin, plus la description de la sémantique de l'application et de son comportement dans le PSM est complète, plus le code généré est complet (Cephas, 2006).

Le passage d'un modèle vers l'autre se fait via des règles de transformation, comme illustré par la Figure. 2.5. Nous distinguons entre les types de transformation suivants (OMG-MDA, 2001) :

- CIM à PIM : cette transformation concerne la génération d'un modèle formel indépendant de la plateforme à partir des spécifications des utilisateurs. Ce type de transformation est généralement effectué manuellement ou semi automatiquement
- PIM à PIM : ce type de transformation consiste en le raffinement d'un modèle PIM. Il concerne l'enrichissement, le filtrage ou la spécialisation d'un modèle PIM, durant le cycle de développement du système, sans la nécessité de détails relatifs à la plateforme.
- PIM (et PDM) à PSM : Cette transformation est utilisée lorsque le PIM est suffisamment raffiné pour être projeté sur une plateforme d'exécution. Cette transformation dépend des caractéristiques de la plateforme cible décrites avec de l'UML ou des profils UML.
- PSM à PSM : c'est aussi un cas de raffinement de modèle. Ce type de transformation est utile pour l'implémentation et le déploiement de composants.
- PSM à Code : contrairement aux transformations précédentes, cette transformation est de type modèle à texte. Elle vise à générer le code source de l'application, de façon totale ou partielle, à partir des modèles PSM de l'application.
- PSM à PIM : cette transformation est nécessaire pour la génération de modèles indépendants de la plateforme à partir d'implémentation existante. Elle est généralement difficile à automatiser. Elle peut cependant être effectuée via des outils.

2.3. Les Promesses de la MDA

La MDA offre plusieurs avantages que nous pouvons résumer comme suit (Kleppe, 2003) :

2.3.1. Productivité

Dans un contexte MDA, les développeurs se concentrent sur le développement des modèles PIMs, au moment où les modèles PSMs et codes sont générés via des transformations PIM à PSM et PSM à code. Bien évidemment, les transformations doivent être préalablement définies, ce qui constitue une tâche difficile et spécialisée et qui doit être effectuée par une personne

hautement qualifiée. Cependant, ces transformations seront définies une seule fois et pourront être appliquées, par la ensuite, pour générer plusieurs systèmes. Un retour sur investissement important est alors tiré de la définition des transformations.

Ainsi, grâce à la MDA, la productivité est améliorée. En effet, d'une part, les développeurs ont moins de charge de travail car ils n'ont pas besoin de concevoir ou d'implémenter les détails relatifs à la plateforme, car ces détails sont déjà traités dans la définition de la transformation. D'une autre part, les développeurs pourront ainsi accorder plus d'attention à la résolution des problèmes métier. Cela se traduit par un système qui correspond beaucoup mieux aux besoins des utilisateurs finaux qui obtiennent de meilleures fonctionnalités en moins de temps.

2.3.2. Portabilité

Dans la MDA, la portabilité est assurée à travers la focalisation sur les modèles PIMs, qui sont par définition indépendants de la plateforme. Le même PIM peut être automatiquement transformé en plusieurs PSMs. Tout ce qui est défini au niveau du PIM est alors complètement portable. Ce qui est le principe de base de la MDA « model once, generate everywhere ».

2.3.3. Interopérabilité

Dans une architecture MDA, les PSMs générés depuis un même PIM peuvent être liés, ceci est appelé le pont (bridge). En effet, étant capable de générer plusieurs PSMs à partir d'un PIM, signifie que toutes les informations nécessaires pour relier les PSMs sont disponibles. Pour chaque élément dans un modèle PSM, nous savons depuis quel élément du modèle PIM il a été généré et de même, nous savons vers quel élément du second PSM cet élément du PIM à été transformé. Nous pouvons ainsi déduire comment les éléments du premier PSM peuvent être reliés aux éléments du deuxième PSM, comme illustré par la Figure 2.6. Ceci est le sens même de l'interopérabilité.

Grace à cette interopérabilité, la MDA permet également de remédier à l'obsolescence technologique faisant que des outils encore fonctionnels deviennent obsolètes parce qu'ils ne sont plus compatibles avec de nouveaux outils. De cette manière, les organisations pourront profiter plus longtemps de leurs investissements dans les applications logicielles.

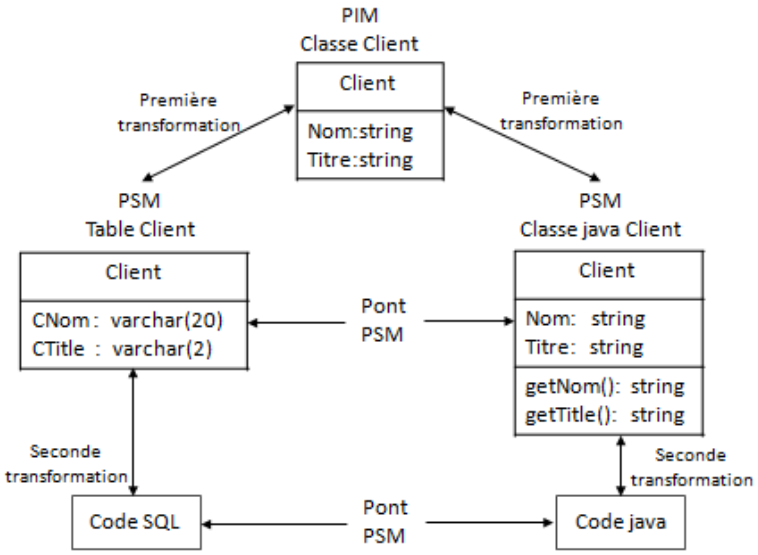


Figure 2.6 L'interopérabilité dans la MDA (Kleppe, 2003)

2.3.4. Maintenance et documentation

En adoptant une démarche MDA les développeurs se concentrent sur le modèle PIM. Ce dernier est utilisé pour générer le modèle PSM, qui à son tour est utilisé pour générer le code. Le modèle PIM est alors une représentation exacte du code. Il représente ainsi la documentation de haut niveau pour tout système. La seule différence est que le PIM n'est pas abandonné une fois écrit, mais il est impérativement mis à jour pour tout changement afin de permettre la génération du PSM et code associés. D'une autre part, la maintenance des applications développées suivant une architecture MDA est réduite, du fait que tout nouveau besoin est facilement intégré. Ceci permet une réduction importante du coût total du projet, sachant que la maintenance d'un projet constitue près de 90% de son coût global (Prakash et al, 2018).

2.4. Les standards de modélisation de l'OMG

UML (Unified Modeling Language) : est un langage de modélisation, visualisation et documentation des applications logicielles. Il est le langage le plus utilisé dans la méta-modélisation, en particulier dans les architectures MDA. Grâce à son concept de base, classe/association, l'UML offre un mécanisme d'extension, appelé profil, permettant d'étendre le vocabulaire d'UML pour s'adapter à un domaine ou plateforme spécifique (Gaševic et al, 2006). Un profil est défini en utilisant des stéréotypes, tags et contraintes appliqués à des éléments spécifiques du modèle, comme les classes, les attributs, les opérations et les activités. Un stéréotype sert en fait à créer de nouveaux éléments pour un domaine spécifique. Il est représenté en plaçant le nom du stéréotype entre guillemets. Un tag ou valeur tagguée, sert à

définir de nouvelles propriétés pour un stéréotype ou élément existant du modèle, sous forme de paire tag-valeur placée entre accolades. Enfin, les contraintes sont utilisées pour étendre la sémantique UML en ajoutant de nouvelles règles ou en modifiant des règles existantes. Une contrainte peut être spécifiée en langage naturel ou en OCL (Object Constraint Language).

BPMN (Business Process Model and Notation): il a pour rôle d'offrir une notation standard, claire et compréhensible des processus métier aux experts du domaine ayant créé ces processus, ainsi qu'aux développeurs responsables de leur implémentation, mais aussi aux utilisateurs responsables de la gestion et le suivi de ces processus. Le BPMN propose une représentation graphique permettant aux entreprises de comprendre leurs processus et de les communiquer via un standard. Il offre également la possibilité de traduire les langages de système BPM (Business Process Management), comme le WSBPEL (Web Service Business Process Execution Language), en une représentation graphique et vice-versa (OMG-BPMN, 2013). Le BPMN propose cinq catégories d'éléments :

1. **Flux d'objets** : ils servent à définir le comportement d'un processus métier. Ils sont composés de :
   - **Evénements** : ils se produisent durant l'exécution du processus et ont, généralement, un impact sur celui-ci. On différencie entre événements de début, intermédiaires et de fin.
   - **Activités** : se sont les travaux effectués dans une entreprise. Elles peuvent être atomiques ou composées. Dans un processus, une activité est un sous processus (activité composée) ou une tâche (activité atomique).
   - **Branchements** (Gateways) : ils sont utilisés dans un processus pour contrôler la divergence ou la convergence des flux (union, jointure etc.).
2. **Données** : fournissent des informations sur les données nécessaires à une activité pour être effectuée, et/ou les données qu'elle produise.
3. **Objets de connexion,** contiennent :
   - **Flux de séquence** : ils sont utilisés pour indiquer l'ordre d'exécution des activités.
   - **Flux de message** : ils servent à définir les messages échangés entre participants.
   - **Associations** : elles servent à relier des informations ou artefacts avec les éléments graphiques du modèle BPMN.
4. **Couloirs** (Swimlanes) : ils servent à regrouper les éléments d'un processus. Ils peuvent être de type :
   - **Piste** (Pool) : elle permet d'isoler un ensemble d'activités du reste, sous forme d'un processus exécutable. elle peut, également, être vide sous forme d'une boite noire.

- **Corridor** (Lane) : c'est une sous partition au sein d'un processus ou même au sein d'un pool. Il est utilisé pour organiser et catégoriser les activités.

5. **Artefacts** : ils servent à fournir plus d'informations sur les processus. Un artefact peut être :
   - **Groupe** : est un groupement d'éléments graphiques et qui n'affecte pas le flux de séquence.
   - **Annotation** : est le mécanisme permettant au concepteur de fournir plus d'informations aux lecteurs ou aux utilisateurs du diagramme BPMN.

2.5. Standards de modélisation décisionnelle

Il existe deux standards de modélisation des projets décisionnels : le CWM (Common Warehouse Metamodel), proposé par l'OMG en septembre 1999 (CWM, 2003) et l'OIM (Open Information Model), proposé en juillet 1999 par la MDC (Meta Data Coalition), une organisation indépendante dédiée à la création de standards et de modèles de métadonnées pour l'échange d'informations entre les systèmes hétérogènes (MDC-OIM, 1999). Les deux standards sont indépendants des éditeurs et sont basés sur le langage UML pour la méta-modélisation et l'XML pour l'échange de données entre les systèmes. Le CWM offre 22 métamodèles correspondant chacun à un aspect du projet décisionnel, classifiés en cinq catégories : management, analyse, ressource, fondement et modèle d'objet (voir Figure 2.7).

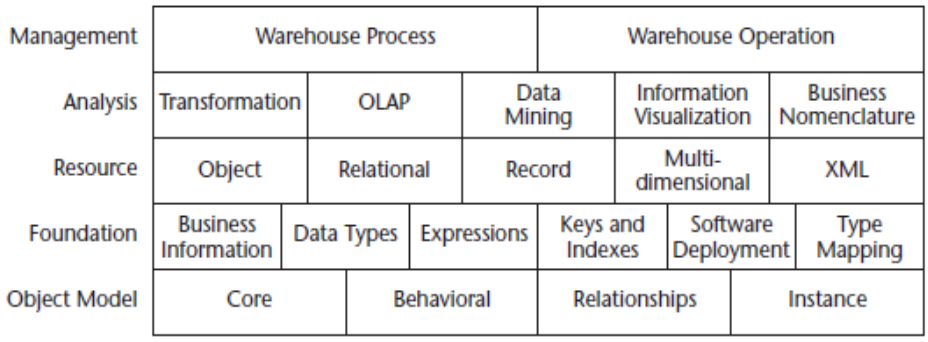


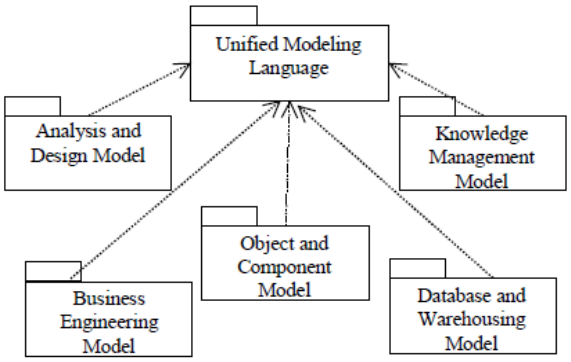
Figure 2.7 Métamodèles CWM (CWM, 2003)

Quant à l'OIM, il propose cinq modèles contenant chacun un ensemble de sous-modèles comme le montre la Figure 2.8.

Ils existent, cependant, des différences entre les deux standards. La différence fondamentale concerne le schéma OLAP, sujet de notre recherche. En fait, le métamodèle CWM est un modèle purement sémantique qui décrit les concepts OLAP généraux, sans dresser aucun aspect de déploiement logique ou physique. Contrairement à l'OIM, qui offre un ensemble de méta-classes

relatives au déploiement physique des cubes OLAP, comme les classes : OLAP server, Deployed OLAP Server, Data Source et Connection.

.



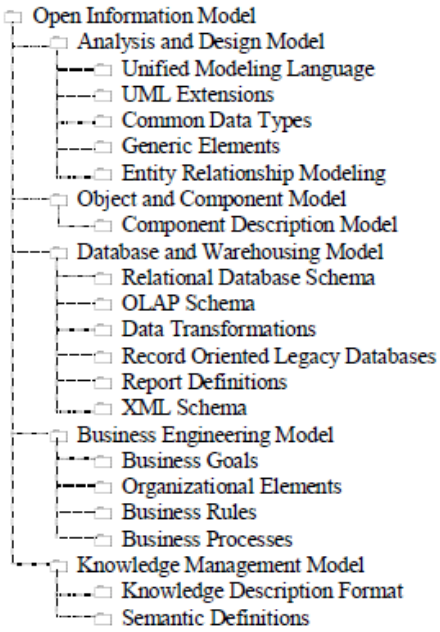


Figure 2.8 Packages de l'OIM (MDC-OIM, 1999)

Une deuxième différence entre les deux standards concerne les modes de déploiement ; en effet, le métamodèle CWM-OLAP définit des concepts généraux qui peuvent être mappés vers un modèle multidimensionnel ou relationnel correspondant respectivement aux deux modes de déploiement MOLAP et ROLAP. Le métamodèle OIM-OLAP, par contre, regroupe ces deux concepts. Il permet de définir un mode de déploiement pour chaque cube ou partition.

Ceci justifie notre choix des métamodèles OIM au lieu du CWM dans notre approche MDA visant à automatiser la génération du code d'implémentation à partir des modèles physiques.

2.6. La MDA dans les systèmes décisionnels

Afin d'automatiser le processus de mise en place des projets décisionnels, l'architecture MDA a été largement utilisée dans la littérature, vu tous ses avantages, notamment la séparation des spécifications fonctionnelles de ceux techniques en plus de la portabilité, l'interopérabilité et la réutilisabilité des modèles. Parmi les travaux ayant adopté cette architecture on cite :

Lujan Mora et al: ont proposé un ensemble de profils UML qui décrivent les concepts OLAP (Lujan - Mora et al, 2006), notamment les faits, attributs de fait (mesures), dimensions et attributs de dimension. Le métamodèle proposé traite également le type de relation plusieurs-à-plusieurs entre fait et dimension via les faits dégénérés (degenerate fact). En outre, le métamodèle inclut les dimensions dégénérées (dimension contenu dans le fait) et les hiérarchies.

Les auteurs ont établi, également, un ensemble de directives pour obtenir le modèle multidimensionnel à travers une architecture à trois niveaux : package, fait et dimension. Enfin, dans le cadre d'une architecture MDA (Model Driven Architecture), les auteurs ont proposé un ensemble de transformations QVT (Query/View/Transformation) pour la génération automatique du modèle relationnel à partir du modèle conceptuel proposé. Ce travail deviendra la base de nombreuses approches futures.

Zepeda et al : dans leurs travaux (Zepeda et al, 2008, 2010, 2015), les auteurs ont proposé une méthode mixte orientée but pour la génération du modèle multidimensionnel. L'approche consiste à générer plusieurs modèles relationnels à partir des schémas entité/association des bases de données opérationnelles. Ensuite, sélectionner le modèle adéquat qui s'adapte aux besoins des utilisateurs. Pour modéliser ces derniers, les auteurs ont opté pour le graphe Goal Refinement Tree (GRT) permettant, en plus de la modélisation des objectifs business, de décomposer ces derniers en sous objectifs jusqu'à l'obtention de tâches tangibles. A partir de ces tâches, l'ensemble des informations nécessaires à l'analyse ou l'achèvement de ces tâches est déduit. Pour ce faire, les auteurs ont utilisé le diagramme d'activité afin de tracer chaque tâche et les informations qu'elle requière ou qu'elle produise. Les auteurs ont dressé aussi un ensemble de règles de transformation pour obtenir le modèle multidimensionnel à partir du modèle relationnel obtenu précédemment. Enfin, les auteurs génèrent le code SQL nécessaire à la création du DW relationnel.

Mazon et al : à travers leurs nombreux travaux sur les systèmes décisionnels, les auteurs ont proposé une approche orientée but qui distingue entre trois catégories de buts selon leurs niveaux d'abstraction : buts stratégiques, buts de décision et buts d'information. Pour les modéliser, les auteurs proposent l'utilisation du Framework iStar (i*) (Mazon et al, 2007) via ses deux modèles : SD (Strategic Dependency), qui décrit les relations de dépendance entre les acteurs et le modèle SR (Strategic Rational), qui fournit une vue détaillée des besoins utilisateurs. Les auteurs ont proposé par la suite une approche de mise en place des DSS basée sur l'architecture MDA (Mazon et al, 2008). L'approche proposée divise le projet BI en cinq composants, à savoir : sources de données, ETL, multidimensionnel (DW), personnalisation et application, auxquels l'approche MDA peut être appliquée séparément. Les auteurs ont ensuite détaillé l'application de l'approche MDA pour la génération du DW (Mazon et al, 2009). L'approche proposée consiste à générer un model multidimensionnel (PIM), à partir des besoins utilisateurs, représentés avec le model i*, ensuite, réconcilier le modèle obtenu avec les schémas des sources de données de manière semi-automatique. A partir du modèle multidimensionnel obtenu, le modèle relationnel du DW ainsi que le modèle multidimensionnel pour l'OLAP sont générés.

Pour ce faire, les auteurs ont utilisé le profil UML proposé par Lujan Mora (Lujan Mora et al, 2006) pour la modélisation du PIM et le standard CWM pour la représentation des modèles relationnel et OLAP. Les auteurs fournissent également une description des règles de transformation requises à l'obtention du modèle relationnel ainsi que la génération du code SQL associé.

Pardillo et al : dans leur approche (Pardillo et al, 2008), les auteurs ont proposé la génération automatique des deux modèles relationnel et OLAP à partir du model conceptuel dans un contexte MDA en utilisant le standard CWM pour la modélisation des deux modèles relationnel et OLAP. Les auteurs ont dressé également l'ensemble des règles de transformation requises. Toutefois, la génération du code final n'est pas traitée.

Soler et al : toujours dans le cadre d'une approche MDA, les auteurs (Soler et al, 2009) ont choisi de traiter l'aspect sécurité dans les data warehouses relationnels. L'approche proposée comprend une solution de modélisation des data warehouses sécurisés, en plus de l'ensemble des règles de transformation requises pour la génération du DW relationnel, ainsi que le code SQL de création des tables associées et des contraintes de sécurité.

Blanco et al : à travers leurs travaux (Blanco et al, 2015a, 2015b), les auteurs ont continué l'approche de Soler et al en intégrant, cette fois-ci, la couche OLAP. L'approche proposée consiste à modéliser l'aspect sécurité en étendant le standard CWM. A partir de ce modèle, générer les fichiers système du cube OLAP relatifs à la plateforme SQL Server Analysis Services (SSAS), sous forme de fichiers XML .dim, .cube, mais également fichiers .role qui décrivent les permissions relatives au cube et aux dimensions.

Neil et al : une fois encore les auteurs Neil et al, ont opté pour l'architecture MDA pour gérer, cette fois-ci, l'historisation dans les DWs (Neil et al, 2014). L'approche proposée consiste à modéliser et stocker les structures temporelles, permettant de gérer les attributs de dimension variant dans le temps. Les auteurs ont traité également la génération du code SQL pour la création du DW relationnel. Toutefois, l'exploitation de cet historique dans le modèle OLAP n'a pas été traitée.

Maté et al : dans leur travail (Maté et al, 2014), les auteurs traitent le problème de taille et de lisibilité des schémas i* pour la représentation des besoins utilisateurs, ainsi que la gestion des changements, chose dont souffrent toutes les approches précédentes. Pour ce faire, les auteurs proposent une extension du schéma i*, en intégrant la modularité dans ce dernier. Ainsi, les auteurs introduisent trois modules : décision, information et hiérarchie, permettant de packager les buts ou leurs détails dans des modules, pour cacher les détails et faciliter ainsi la lecture et la

modification. Enfin, les auteurs dressent un ensemble de lignes directives pour utiliser les modules proposés.

Atigui et al : les auteurs proposent une méthode mixte et semi automatique de modélisation et d'automatisation des DWs, basée sur l'architecture MDA (Atigui et al, 2011). Le Framework proposé consiste à générer le modèle conceptuel du DW à partir des besoins utilisateurs et schémas des sources de données qu'ils obtiennent à partir de leurs schémas physiques (reverse engineering), un traitement que les auteurs décrivent brièvement. Les auteurs proposent pour cela un profil UML pour représenter le modèle conceptuel du DW. Ensuite, à partir de ce modèle, un modèle logique (relationnel) est généré via un ensemble de règles de transformation automatiques, ainsi qu'un modèle physique, que les autres représentent par les vues matérialisées ORACLE. Notant aussi que le modèle conceptuel proposé ignore les relations de type plusieurs-à-plusieurs.

2.7. Analyse des travaux de l'état de l'art

Afin de récapituler, comparer et analyser les travaux de l'état de l'art, concernant l'élaboration des systèmes décisionnels via une architecture MDA, nous avons établi des critères de comparaison relatifs à chaque phase du cycle de conception du DSS. Le tableau 2.1 récapitule et synthétise l'analyse de ces travaux selon les critères retenus:

- Type de la démarche adoptée : Dirigée par les données (Données), dirigée par les besoins utilisateurs (BU), dirigée par les buts (Goal) ou mixage des besoins utilisateurs et des sources de données (Mixte)
- Modélisation des exigences de haut niveau (ER)
- Modélisation des exigences de bas niveau (LR)
- Passage ER à LR : peut être automatique(A), semi-automatique (SA) ou manuelle (M)
- Restriction des relations dans les modèles ER et LR
- Représentation du modèle PIM (conceptuel)
- Traitement de la relation plusieurs-à-plusieurs dans le modèle PIM
- Passage CIM à PIM : peut être automatique(A), semi-automatique (SA) ou manuelle (M)
- Représentation du modèle relationnel
- Représentation du modèle OLAP
- Traitement de la relation plusieurs-à-plusieurs dans les modèles PSMs
- Passage PIM à PSM : peut être automatique(A), semi-automatique (SA) ou manuelle (M)
- Description de la plateforme cible
- Implémentation relationnelle

- Implémentation OLAP
- Méthode de génération du code final : peut être automatique(A), semi-automatique (SA) ou manuelle (M)
- Collecte des métadonnées pour la traçabilité et la documentation du système: peut être automatique(A), semi-automatique (SA) ou manuelle (M)

Le caractère «-» désigne que l'aspect en question n'a pas été traité.

Tableau 2.1 Récapitulatif des travaux de l'état de l'art

Critères de comparaison	**Lujan Mora et al**	**Mazon, Pardillo et al**	**Zepeda et al**	**Blanco et al**	**Neil et al**	**Atigui et al**	**Soler et al**
Type de démarche	BU	Goal	Mixte	BU	Données	Mixte	BU
Niveau CIM							
Modélisation ER	-	Modèle SD	graphe GRT	-	-	-	i*
Modélisation LR	-	Modèle SR	Diagramme d'activité	-	-	-	-
Passage ER à LR	-	SA	M	-	-	-	-
Restriction des relations	-	Oui	Non	-	-	-	-
Niveau PIM							
Métamodèle	UML	UML	MD CWM	UML	Graphe temporel	UML	UML
Traitement de la relation plusieurs-à-plusieurs	Oui	Oui	Non	Non	Non	Non	-
Passage CIM à PIM	-	SA	A	-	A	M	-
Niveau PSM							
Modèle Relationnel	-	CWM	-	-	CWM	E/A	CWM
Modèle OLAP	-	MD CWM	-	Secure OLAP-CWM	-	-	-
Traitement de la relation plusieurs-à-plusieurs	-	-	-	-	-	-	-
Passage PIM à PSM	-	A	-	A	A	A	A
Description de la plateforme cible	-	-	-	-	-	-	-
Implémentation							
Relationnelle	SQL	SQL	SQL	SQL	SQL	-	SQL
OLAP	-	-	-	Fichiers XML	-	-	-
Génération du code final	A	A	A	A	A		A

Collecte des Métadonnées et documentation du système	-	-	-	-	-	-	-

2.8. Conclusion

Etant donné tous ses avantages, l'architecture MDA est l'une des approches les plus utilisées dans la littérature pour la conception et la mise en place des projets décisionnels. En effet, en plus de l'automatisation du processus de développement, la MDA garantit la portabilité et l'interopérabilité des applications en plus de l'amélioration de la productivité des développeurs et la réduction des tâches de maintenance. Elle permet ainsi de réduire le coût et le délai de mise en place et de maintenance des projets.

On constate, toutefois, l'absence d'une approche globale qui traite tous les aspects de mise en place d'un DSS. En effet, au niveau de la définition des besoins utilisateurs, la description des sources de données dans toutes les méthodes proposées, n'est pas intégrée dans le modèle CIM. La réconciliation avec les sources de données se fait alors en postériori, ce qui peut entraîner une révision des besoins utilisateurs après leur validation. De plus, les méthodes utilisées, comme i* et GRT, présentent l'inconvénient de complexité et d'illisibilité une fois les besoins utilisateurs s'élargissent. Ajoutant aussi l'absence de mécanisme de contrôle de bonne forme du modèle conçu. D'une autre part, au niveau de la phase de modélisation logique et physique, on constate que toutes les approches proposées se sont focalisées sur le modèle relationnel du DW, en générant le schéma des tables ainsi que le code SQL relatif à la création de ces dernières, alors que le modèle OLAP a été négligé. Le seul travail ayant traité l'implémentation des cubes OLAP a traité, cependant, la génération des fichiers système relatifs au cube, dimensions et rôles au lieu de générer le code d'implémentation. De plus, les fichiers générés ne contiennent pas la description relationnelle nécessaire à la création des composants OLAP comme la table, la colonne et la taille associées à chaque attribut de dimension. On cite également l'absence de modèle décrivant la plateforme OLAP cible pour permettre le passage automatique du PSM vers le code d'implémentation ainsi que la migration rapide d'une plateforme vers une autre. On remarque aussi que la traduction de la relation plusieurs-à-plusieurs au niveau du modèle logique n'a pas été traitée. Enfin, à l'exception de l'approche proposée par (Jovanovic et al, 2014) (voir chapitre 1), toutes les autres approches n'ont pas traité la collecte et la sauvegarde des métadonnées pour assurer la documentation et la traçabilité dans les DSS.

Pour remédier à tous ces problèmes, nous proposons une approche de modélisation et de mise en place des systèmes décisionnels, basée sur l'architecture MDA. Notre approche mixte, orientée buts, a pour objectif de fournir les outils permettant d'assister les décideurs à définir leurs besoins en termes d'analyse par rapport à leurs buts stratégiques. Elle a pour but aussi, de surpasser les verrous technologiques et les outils BI représentant des boites noires, grâce à l'automatisation du processus de développement en allant jusqu'à la génération du code final. Notre approche traite aussi l'exploitation des modèles de l'architecture MDA pour la collecte des métadonnées et la documentation du DSS.

Chapitre 3

Notre Architecture Dirigée par les Modèles pour les systèmes OLAP

Résumé : Dans les chapitres précédents, nous avons présenté les contraintes relatives à la conception des systèmes décisionnels ainsi que les approches proposées dans l'état de l'art pour la mise en place de ces derniers. En effet, vu leur complexité, taille et évolutivité, l'élaboration des systèmes décisionnels peut s'avérer un processus énormément long et coûteux, en particulier dans le cas d'une mauvaise conception. Ainsi, l'automatisation du cycle de développement de tels projets peut garantir un gain considérable en termes de coût et de délai. Dans ce chapitre, nous présentons notre approche OMDA, semi-automatique et orientée but pour la conception des systèmes OLAP, allant de la collecte et la formalisation des besoins utilisateurs jusqu'à la génération du code d'implémentation de l'entrepôt de données relationnel et du cube OLAP associé. Notre approche traite aussi la collecte et la documentation des métadonnées pour garantir la traçabilité et la gouvernance des données. Enfin, notre proposition, grâce aux nombreux avantages de l'architecture MDA, dont la portabilité, l'interopérabilité et la réutilisabilité des modèles, vise à automatiser le cycle de développement des systèmes décisionnels mais également, à surpasser les verrous technologiques, qui conditionnent le développement des DSS. Ainsi, rendre possible la migration de ces systèmes d'une plateforme vers une autre sans à avoir à refaire la conception ni l'implémentation.

3.1. Introduction

Notre approche basée sur l'architecture MDA, vise à fournir une démarche et des outils permettant de concevoir et d'implémenter les systèmes décisionnels et d'automatiser leur cycle de développement (Letrache et al, 2017). La première étape de notre approche correspond au niveau CIM (El Beggar et al, 2017). Elle consiste à assister les décideurs à définir d'abord, leur besoin exprimé sous forme de buts stratégiques et qui représentent les exigences de haut niveau. Pour les modéliser, nous proposons un profil du diagramme de cas d'utilisation, que nous avons nommé GoalCases (El Beggar et al, 2016). Ce dernier, grâce à un ensemble de stéréotypes personnalisés, permet de modéliser les buts stratégiques ainsi que les objectifs décisionnels requis à l'achèvement des ces buts. Il permet de représenter également les métriques nécessaires à mesurer l'accomplissement de ces objectifs. Par la suite, nous proposons un profil étendant le métamodèle BPMN, appelé SGAP (Strategic Goal Analysis Process), servant à détailler les buts et objectifs modélisés dans le GoalCases afin de déduire, progressivement, les informations et données requises pour analyser et vérifier l'achèvement de ces buts stratégiques.

La deuxième étape de notre approche consiste à déduire le modèle conceptuel (PIM) du nouveau système à partir du modèle CIM, plus précisément, modèle SGAP. Pour cela, nous proposons un profil UML, composé d'un ensemble de stéréotypes représentant les concepts multidimensionnels (Letrache et al, 2017).

Ensuite, nous proposons un métamodèle PDM décrivant les plateformes OLAP. En instanciant ce métamodèle selon la plateforme cible, nous pouvons automatiser le passage du modèle PIM au modèle OLAP et, par la suite, au code final.

A partir du modèle PIM, le modèle relationnel du futur DW/DM est généré automatiquement. Nous générons également le modèle OLAP du cube associé, en utilisant cette foi-ci le modèle PIM en plus du modèle PDM relatif à la plateforme cible. Pour représenter ces deux PSMs, relationnel et OLAP, nous avons opté pour les métamodèles proposés par le standard OIM, dont nous avons justifié le choix dans la section 2.5. Pour générer automatiquement les modèles PSMs, nous définissons un ensemble de règles de transformation de type modèle à modèle.

Enfin, à partir des deux modèles, relationnel et OLAP, le code d'implémentation du DW relationnel et du cube OLAP sont automatiquement obtenus via des règles de transformation de type modèle-à-texte (voir Figure 3.1).

D'autre part, outre la conception des DSS, notre approche OMDA inclut la collecte et la documentation des métadonnées. En effet, un des problèmes courants dans les projets décisionnels est l'absence de documentation et de traçabilité à cause de l'évolution rapide et

continue des DSS. Ceci impacte la stabilité de ces derniers et rend difficile la gestion des changements. Pour cette raison, notre approche intègre la collecte des métadonnées disponibles dans chaque niveau de notre architecture, pour servir par la suite à générer la documentation du système et tracer la lignée des données (data lineage).

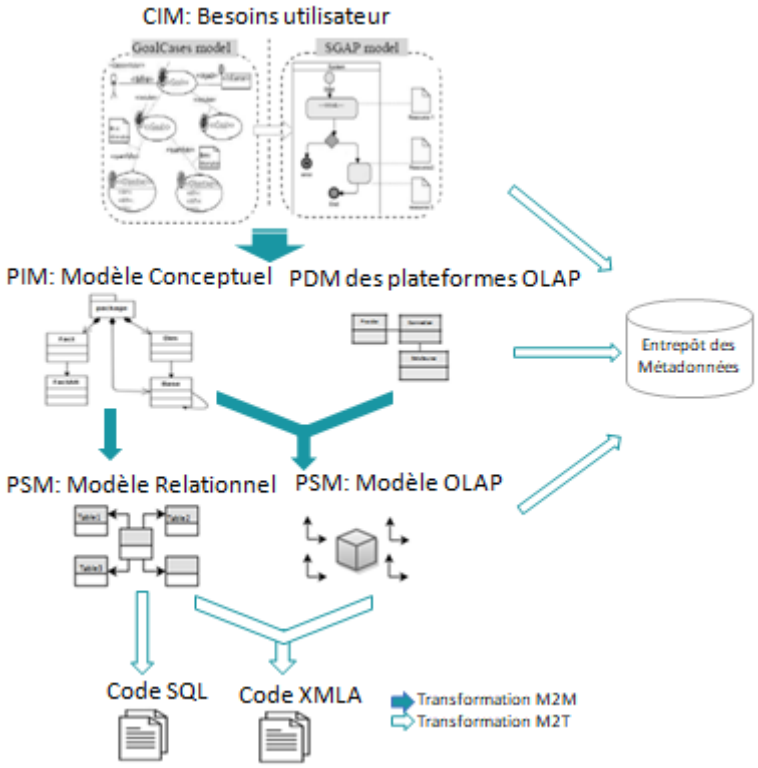


Figure 3.1 Notre Approche OMDA pour l'élaboration des systèmes décisionnels

3.2. Modèle CIM

Dans notre approche OMDA orientée but, le modèle CIM a pour objectif de fournir une démarche et un outil d'échange et de communication simple, compréhensible et fiable, permettant d'assister les décideurs à définir leurs buts stratégiques et aux concepteurs et développeurs BI de décomposer ces buts jusqu'à déduire les métriques que le DW ou data mart (DW/DM) doivent permettre d'analyser.

Pour réaliser ce modèle, nous proposons deux profils permettant de représenter les concepts décisionnels et modéliser les besoins des utilisateurs en deux étapes. La première, consiste à modéliser les exigences de haut niveau, exprimées sous forme de buts stratégiques et objectifs décisionnels, en utilisant notre profil GoalCases, étendant le diagramme de cas d'utilisation UML. La deuxième étape, vise à détailler les buts et objectifs du GoalCases pour en sortir, les éventuels mesures et dimensions qui constitueront les composants du futur modèle PIM. Pour cela, nous utilisons notre profil SGAP qui étend le métamodèle BPMN.

3.2.1. Exigences de haut niveau

La première étape de notre approche consiste à modéliser les exigences de haut niveau via notre profil *GoalCases* qui étend le diagramme de cas d'utilisation UML. La Figure 3.2 présente les différents éléments du GoalCases, composé d'un ensemble de stéréotypes, en plus du métamodèle du diagramme de cas d'utilisation que nous étendons. Dans ce profil, le stéréotype « OrganizationUnit» étend la classe *Acteur* et distingue entre deux rôles, le « DecisionMaker» et le «Influencer». Le premier est responsable de la définition des buts et objectifs décisionnels nécessaires à l'achèvement de ces buts. Alors que le « Influencer » représente un acteur pouvant nuire ou contribuer à l'achèvement des buts. Ainsi, pour différencier entre influencer positif et celui négatif, l'attribut « IsPosetive » peut être utilisé. D'autre part, un influencer peut être interne à l'entreprise, annoté par « InternalInfluencer » ou externe « ExternalInfluencer ». Enfin, le degré d'impact d'un influencer peut être renseigné via l'attribut « Weight ».

Les stéréotypes « Objective » et « Goal » héritent du « BusinessEnd », qui étend la méta-classe *UseCase*. En effet, un objectif est une finalité métier qui peut être mesurée périodiquement (par mois, année, trimestre etc.). Dans notre profil GoalCases, cette périodicité peut être renseignée via l'attribut « Period » et dont les valeurs possibles correspondent à l'énumération « Time ». Par ailleurs, un objectif doit être mesurable, ses métriques correspondent aux mesures de performance du but stratégique que l'objectif doit quantifier. Notre profil GoalCases permet d'exprimer cette dépendance entre but et objectif via le stéréotype « quantifyBy » étendant la méta-classe *Extend*. Le « quantifyBy » est défini par un nom, une unité, une valeur actuelle et un type de métrique. Ainsi, notre modèle différencie entre deux types de métriques; mesures et indicateurs clés de performance (KPI). Les métriques fonctionnent comme des jauges permettant de déterminer la situation d'un objectif via le stéréotype « GoalSituationPoint » qui étend la méta-classe *ExtensionPoint* et qui permet de définir les trois états possibles pour un but : achèvement, avertissement et échec ainsi que les seuils déterminant chaque état. Enfin, la méta-classe *Include* permet de décomposer un but en sous-buts. Toutefois, la décomposition ne peut pas se faire entre buts et objectifs. Ainsi, un but ne peut pas être décomposé en sous-objectifs et de la même manière un objectif ne peut pas contenir des sous buts.

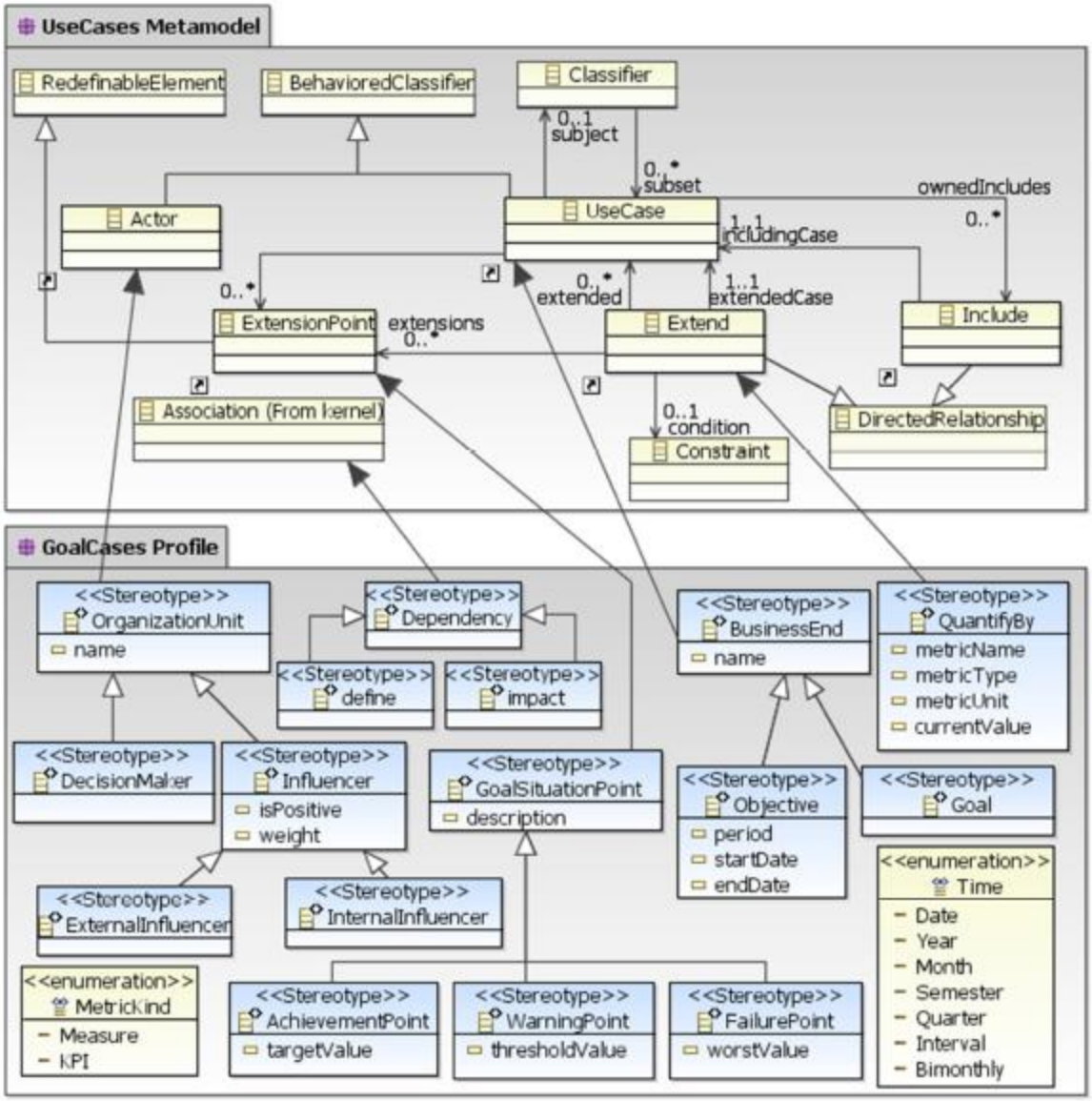


Figure 3.2 Notre profil GoalCases pour la représentation des exigences de haut niveau

3.2.2. Exigences de bas niveau

Une fois les exigences de haut niveau sont définies, la deuxième étape de notre approche est de détailler ces dernières pour obtenir les exigences de bas niveau. Pour ce faire, notre profil BPMN, le SGAP (Figure 3.3), permet une représentation graphique de ces exigences. Ce profil a pour objectif de modéliser les besoins relatifs au nouveau DW/DM sans se préoccuper des contraintes métier ou système.

La réalisation d'un modèle SGAP se fait via une approche descendante, allant des buts stratégiques définis dans le GoalCases vers les objectifs, informations puis données que le DW/DM doit fournir.

Ainsi, on commence dans le SGAP par modéliser le plus haut niveau du processus, représenté par les buts stratégiques et leurs sous-buts s'ils existent, en utilisant la classe « Goal ». Ensuite,

et via la classe « Objective », on définit pour chaque but ou sous buts, les objectifs permettant d'accomplir ce dernier. En outre, les instances d'objectifs dans les modèles SGAP doivent être marquées par un petit indicateur de boucle pour spécifier leur caractère cyclique, en plus d'un indicateur de temps pour préciser la période sur laquelle l'analyse de ces objectifs s'étende.

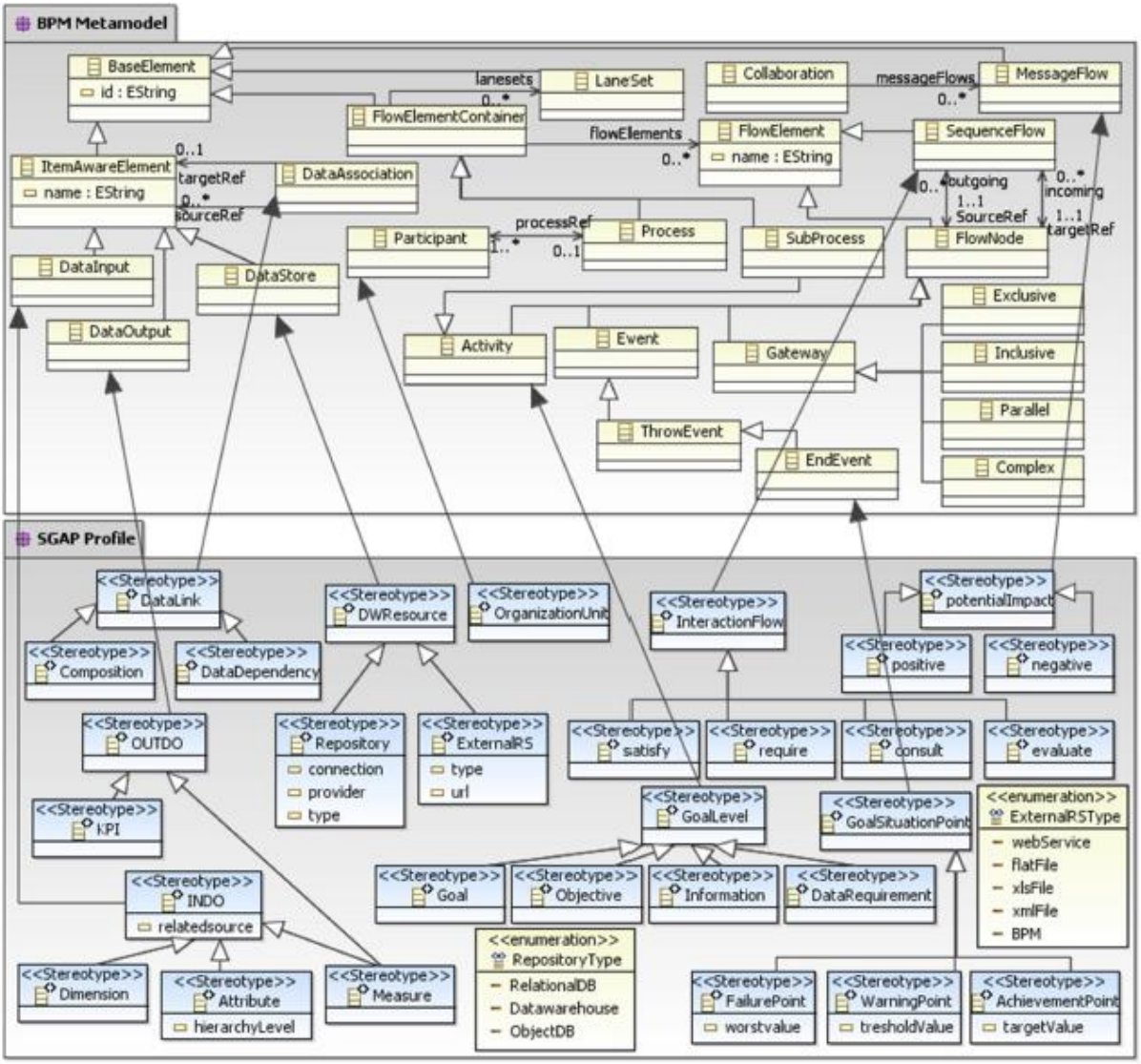


Figure 3.3 Notre profil SGAP pour la représentation des exigences de bas niveau

Le deuxième niveau de but est représenté par le stéréotype « Information ». Il sert à déterminer les informations requises à l'analyse des objectifs prédéfinis. Enfin, le niveau le plus bas, représenté par « DataRequirement », permet de définir les données en sortie « OUTDO », comme les mesures représentées par le stéréotype « Measure » ou KPIs représentés par « KPI », en plus des données en entrée « INDO », représentant les axes d'analyse, à savoir les dimensions « Dimensions » et attributs de dimension « Attribute ». Notant que la classe « OUTDO » étend la méta-classe *DataOutput* et la classe « INDO » étend la classe *DataInput*.

D'autre part, le stéréotype « DWRessource », étendant la méta-classe *DataStore*, sert à identifier les sources de données requises pour la mise en place du DW/DM, et le stéréotype « OrganisationUnit » sert à représenter les acteurs ou responsables des buts. En plus du stéréotype « GoalSituationPoint » étendant la méta-classe *EndEvent*, et qui sert à déterminer l'état du but.

Enfin, la classe « Attribute » est reliée à la classe « Dimension » à travers une relation de type « Composition », alors que les classes « Dimension », « Measure », « KPI » et « DWRessource » sont reliées à la classe « DataRequirement » via une relation de type « DataDependency ». Les deux classes « Composition » et « DataDependency » étendent la méta-classe *DataAssociation.*

3.2.3. Notation, contraintes et directives de réalisation des profils proposés

Dans ce paragraphe nous présentons la notation graphique des éléments des deux profils proposés, ainsi qu'un ensemble de contraintes OCL que nous avons définies pour vérifier les règles de forme des métamodèles profilés afin d'assurer la conformité de leurs instances.

a. Règles de forme

Les règles de forme sont des restrictions ou des contraintes visant à fournir des modèles fiables et correctes. Les restrictions appliquées à nos profils UML sont définies en langage OCL, que nous résumons dans les contraintes principales suivantes :

OCL1 : Lorsqu'un cas d'utilisation est annoté par le stéréotype «Goal», il doit être quantifié avec un cas d'utilisation annoté par le stéréotype «Objective».

```
context Goal
inv: self.base.usecase.ownedextends->select(self.stereotypes->includes(quantifyBy))
->forAll(e | e.extendedCase.stereotypes ->includes(Objective))
```

OCL2 : Tout cas d'utilisation annoté par le stéréotype «Objective» doit spécifier exactement trois «GoalSituationPoint» : AchievementPoint, WarningPoint et FailurePoint, correspondant respectivement aux états : accomplissement, avertissement et échec.

```
inv: self.base.Extend.extensions.notEmpty() implies
self.base.Extend.extensions->select(self.stereotypes->includes(AchievementPoint))
->size() = 1 and self.base.Extend.extensions->select(self.stereotypes
->includes(WarningPoint))->size() = 1 and self.base.Extend.extensions
->select(self.stereotypes->includes(FailurePoint) )->size() = 1
```

OCL3: Un but ne peut contenir que des sous-buts. De même, un objectif ne peut contenir que des sous-objectifs.

```
context BusinessEnd
inv: self.oclIsKindOf(Goal) implies
self.base.usecase.ownedIncludes->
forAll(i | i.includingCase.stereotypes->includes(Goal) and
i.includingCase.stereotypes->excludes(Objective))
inv: self. oclIsKindOf(Objective) implies
self.base.usecase.ownedIncludes->
forAll(i | i.includingCase.stereotypes->includes(Objective) and
i.includingCase.stereotypes->excludes(Goal))
```

OCL4: Le modèle SGAP doit respecter un flux de séquence précis et rigoureux entre les nœuds via l'utilisation de flux appropriés. Le flux de séquence « satisfy » ne peut relier que les activités annotées par « Goal » et « Objective » pour déterminer la manière d'accomplir un but. Alors que le flux « require » ne peut relier que les « Objective » et « Information ». De même, les « Information » sont reliées aux «DataRequirement» à travers le flux «consult». Enfin, le flux « evaluate » permet de définir la relation entre « DataRequirement » et les états du statut final du but via un branchement exclusif.

```
context InteractionFlow
inv: self.oclIsKindOf(satisfy) implies
self.base.sourceRef-> select(self.stereotypes->includes(Goal)) and
(self.base.targetRef->select(self.stereotypes->includes(Goal)
or self.stereotypes->includes(Objective))
or self.base.targetRef.oclIsKindOf(Gateway))
or self.base.sourceRef-> select(self.stereotypes->includes(Objective)) and
(self.base.targetRef->select(self.stereotypes->includes(Objective))
or self.base.targetRef.oclIsKindOf(Gateway))
inv: self.oclIsKindOf(consult) implies
self.base.sourceRef-> select(self.stereotypes->includes(Objective))
and (self.base.targetRef->select(self.stereotypes->includes(Information))
or self.base.targetRef.oclIsKindOf(Gateway))
inv: self.oclIsKindOf(require) implies
self.base.sourceRef-> select(self.stereotypes->includes(Information)) and
(self.base.targetRef->select(self.stereotypes->includes(DataRequirement))
```

or self.base.targetRef.oclIsKindOf(Gateway))
inv: self.oclIsKindOf(evaluate) implies self.base.sourceRef
-> select(self.stereotypes->includes(DataRequirement)) and
self.base.targetRef.oclIsKindOf(ExclusiveGateway)

b. Notation Graphique

Généralement, les stéréotypes utilisés dans les profils UML peuvent avoir une notation graphique particulière (icônes ou images spécifiques). L'icône ou l'image utilisée peut servir de marqueur supplémentaire à côté de la notation graphique UML standard ou carrément remplacer le symbole UML. Dans notre approche, nous avons choisi de conserver la notation UML utilisée dans le diagramme de cas d'utilisation et celle du BPMN, mais qu'on a enrichis avec des marqueurs et des stéréotypes. Ainsi, dans le modèle GoalCases (Figure 3.4), les buts et objectifs sont représentés par des éclipses, comme dans la notation standard du diagramme de cas d'utilisation, avec en plus, des stéréotypes et icônes appropriés. Pour les décideurs, on les représente avec un bonhomme annoté par « DecisionMaker » et les « Influencer » par un simple rectangle, marqué par les stéréotypes « InternalInfluencer » ou « ExternalInfluencer » selon leur type. Un acteur est alors associé à un but ou un objectif via les liens stéréotypés « define » ou « impact ». Enfin, les propriétés des métriques, à savoir le nom, l'unité et le type, sont placées dans des notes, et de même pour les statuts de but (GoalSituationPoint) qu'on place à l'intérieur des compartiments des objectifs.

D'autre part, dans le modèle SGAP (Figure 3.4), les niveaux de buts correspondent à des activités représentées par des rectangles avec l'icône ou le stéréotype approprié. Pour les données en entrée (INDO) et en sortie (OUTDO) ainsi que les sources de données (Repository), on utilise la notation BPMN standard, à l'exception des sources externes (ExternalRS) qu'on représente par un symbole de document. En outre, et afin de garantir l'utilisation de liens précis et correctement définis, le SGAP fournit des associations spécifiques pour relier les éléments du modèle. Par exemple, on relie les attributs et dimension via des associations de composition, alors que les mesures, KPIs, dimensions et ressources, sont reliés aux « DataRequirement » à travers des associations de dépendance. Ajoutant aussi que les associations annotées par le stéréotype « InteractionFlow » servent à relier différents niveaux de flux. Enfin, les associations annotées par le stéréotype « PotentialImpact » servent à différencier entre impact positif et impact négatif d'un « influencer » sur l'accomplissement d'un but.

c. Directives de réalisation du GoalCases

Un modèle GoalCases est une vue sommaire des exigences du système, il doit donc représenter les buts stratégiques définis par les décideurs, leurs sous-buts s'ils existent, leurs responsables ainsi que les objectifs mesurables permettant d'accomplir ces buts. Pour réaliser ce modèle, nous proposons de suivre les directives ci-dessous :

Directive 1 : Un but stratégique peut être clair et précis, comme il peut être vaste ou complexe, dans ce cas, il peut être décomposé en sous-buts plus simples à analyser. On utilise alors l'association UML « include » pour déclarer la dépendance entre les buts et leurs sous-buts.

Directive 2 : Il est important de différencier entre but ou sous-but et objectif. En fait, un objectif est généralement défini pendant une période. Il est aussi quantitatif plutôt que qualitatif, spécifique plutôt que général et ne peut s'étaler au-delà de son calendrier. Un objectif doit donc indiquer précisément comment le but stratégique peut être accompli. On représente alors la relation entre but et objectif par l'association « quantifyBy ».

Directive 3 : une fois que les buts et objectifs sont identifiés, on définit les acteurs responsables des buts, annotés par « DecisionMaker » et ceux impactant positivement ou négativement l'accomplissement de ces buts, annotés par « Influencer ».

Directive 4 : Pour chaque objectif, on définit les métriques permettant de mesurer le degré de satisfaction de cet objectif décisionnel. Une métrique peut être une mesure ou un KPI. Pour la représenter, on utilise le symbole de note relié par une ligne en pointillés à l'association « quantifyBy ». Bien évidemment, chaque objectif définit les seuils relatifs à ses trois états possibles, à savoir l'accomplissement « AP », l'avertissement «WP » et l'échec « FP », tels que déterminés par le « DecisionMaker ».

d. Directives de réalisation du SGAP

Le SGAP est une vue analytique et détaillée des exigences. Son rôle est de décomposer et détailler, pas à pas, les buts et objectifs définis dans le GoalCases, allant du plus haut niveau des besoins stratégiques, jusqu'au niveau détaillé et très granulaire, formant ainsi un flux de buts, d'objectifs, d'informations et de données requises à l'élaboration du futur DW/DM. Pour le réaliser, nous proposons de suivre les directives ci-dessous :

Directive 1: l'élément racine dans un modèle SGAP est le Pool, une sorte de conteneur servant à cerner un but. Il peut contenir des Lanes associés chacun à un rôle organisationnel. On définit alors pour chaque rôle, les buts et objectifs dont il est responsable. D'autre part, on utilise les

pools pour représenter les « influencer » qu'on relie au pool du but concerné via un flux de séquence annoté par « positive impact » ou « negative impact » selon la nature de l'impact.

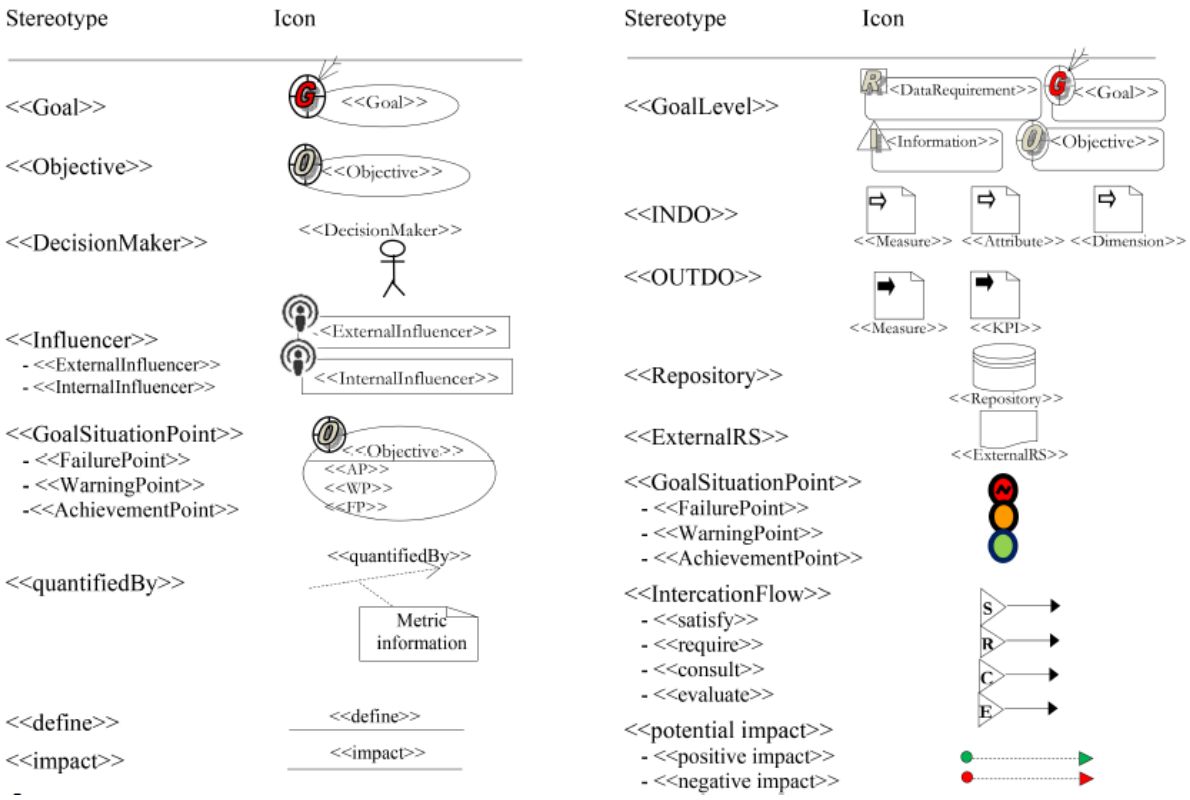


Figure 3.4 Notation graphique des profils GoalCases et SGAP

Directive 2 : l'étape suivante, consiste à représenter les corrélations entre les buts, sous-buts et objectifs, telles que modélisées dans le GoalCases. Ces relations sont représentées à travers différents types de branchements qui précisent la manière dont les buts seront accomplis. Nous utilisons pour cela les notations standards du BPM que nous interprétons dans le contexte de définition des spécifications d'un DW comme suit :

Branchement parallèle : indique qu'un but n'est satisfait que si tous les sous-buts ou objectifs sont satisfaits via l'atteinte des valeurs cibles de leurs métriques.

Branchement exclusif : indique qu'un but n'est satisfait que si l'un de ses sous-buts ou objectifs est atteint.

Branchement inclusif : est utilisé pour fusionner une combinaison de chemins alternatifs et parallèles. Contrairement à la passerelle exclusive, tous les objectifs sont évalués. L'évaluation positive d'un objectif n'annule pas l'évaluation des autres objectifs. Optionnellement, un chemin par défaut peut être défini pour le cas où aucun des objectifs n'a atteint sa valeur cible.

 Branchement complexe : peut être utilisé pour modéliser un comportement de synchronisation complexe entre des objectifs ou sous-buts et le but principal. Il est alors possible de définir des prédicats ou conditions sur les branchements pour représenter des corrélations personnalisées.

Directive 3 : toujours dans le processus de décomposition des buts stratégiques, on précise cette fois-ci les informations requises au suivi des objectifs. Un objectif peut alors exiger plusieurs informations.

Directive 4 : arrivé au plus bas niveau du flux des buts, on précise maintenant, pour chaque information identifiée dans l'étape précédente, les données que le DW doit fournir afin de permettre le suivi des objectifs. On précise alors les trois types de données suivants: les entrées représentant les dimensions et leurs attributs, les sortis représentant les mesures et KPIs et les sources de données opérationnelles nécessaires au chargement de chaque donnée.

Directive 5 : comme dans le GoalCases, la définition des métriques doit être enrichie par une évaluation des résultats de ces dernières afin de déduire le degré de satisfaction du but à analyser. Ceci permettra donc, aux décideurs comme aux analystes, de juger de la pertinence des objectifs et informations définis précédemment.

3.3. Modèle PIM

Le modèle PIM ou modèle conceptuel est le deuxième niveau de notre architecture MDA. Ce dernier est déduit des besoins détaillés des utilisateurs, modélisés avec le profil SGAP. On représente ce modèle conceptuel par un schéma multidimensionnel, généralement en étoile, composé d'un fait relié à un ensemble de dimensions. Pour le modéliser, nous avons étendu le métamodèle proposé par (Lujan-Mora et al, 2006). Ce dernier fournit un profil UML décrivant les principaux concepts multidimensionnels. On retrouve alors le fait, composé d'attributs de fait (mesures) agrégés par dimensions. Le fait est composé aussi de dimensions dégénérées, qui sont des dimensions n'ayant pas d'attributs et qu'on crée alors dans le fait au lieu de les créer dans des dimensions séparées.

D'autre part, les dimensions sont composées de bases représentant chacune un niveau dans une hiérarchie. Chaque base est définie par un identifiant OID et un descripteur, en plus des attributs de dimension. Enfin, le modèle proposé permet de représenter le type de relation plusieurs-à-plusieurs entre fait et dimension via la classe d'association appelée fait dégénéré.

Toutefois, ce modèle permet la représentation d'un seul type de relation plusieurs-à-plusieurs où le fait dégénéré est relié directement au fait. Pour cela, nous avons modifié ce modèle de façon à supporter le cas général de relation plusieurs-à-plusieurs où le fait et fait dégénéré peuvent être reliés directement ou via une dimension partagée. En fait, pour permettre une relation plusieurs-à-plusieurs, le fait dégénéré et le fait principal doivent partager au moins une dimension. Dans le modèle proposé par (Lujan-Mora et al, 2006), le fait et fait dégénéré doivent partager toutes les dimensions du fait principal. Pour remédier à cela, nous utilisons une classe d'association qui définit le type de relation reliant chaque dimension au fait. Dans le cas de relation plusieurs-à-plusieurs, un fait dégénéré (fait intermédiaire) est requis. Ce dernier doit partager au moins une dimension avec le fait principal en plus de la dimension concernée par la relation plusieurs-à-plusieurs. En outre, nous représentons dans notre modèle, les dimensions dégénérées comme type de dimension au lieu de composant du fait et ce via l'attribut IsDegenerate. Nous avons ajouté, enfin, un attribut représentant la fonction d'agrégation relative à chaque mesure (Figure 3.5).

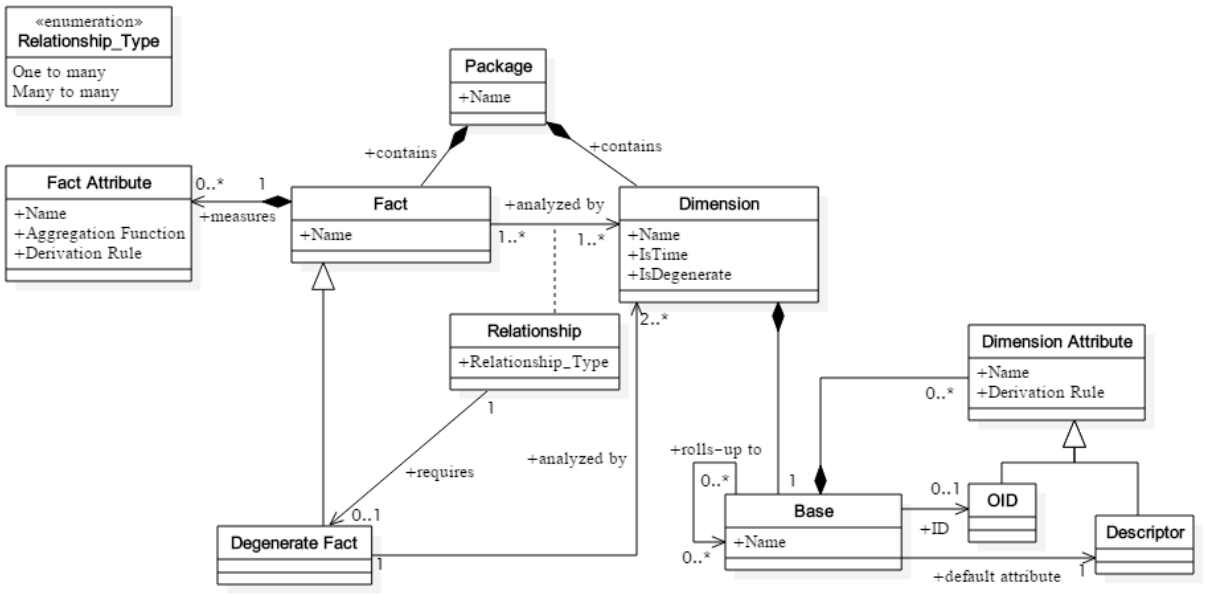


Figure 3.5 Métamodèle multidimensionnel étendu

3.4. Modèle PDM

Dans une architecture MDA, un modèle PDM contient les informations nécessaires à la génération d'un modèle spécifique à une plateforme depuis le modèle indépendant, en décrivant l'architecture technique et les capacités de cette plateforme. Dans notre approche, nous proposons un modèle PDM pour décrire les plateformes OLAP. En effet, contrairement aux plateformes SGBD (Système de Gestion des Bases de Données), les plateformes OLAP diffèrent aussi bien dans le jargon que dans les capacités et options supportées.

Par ailleurs, dans le PDM proposé, une plateforme OLAP permet de manipuler des conteneurs « Container » représentant la racine d'une solution OLAP, comme par exemple, les bases de données dans la plateforme SSAS ou les schémas dans Mondrian. En outre, un conteneur est composé d'objets OLAP comme les cubes, mesures et dimensions (voir Figure 3.6). Un objet OLAP peut avoir une relation avec un autre objet comme mesure avec cube ou niveau avec hiérarchie. De plus, chaque relation peut être décrite par des options comme le type de relation entre fait et dimension qui peut prendre les valeurs « one-to-many » ou « many-to-many ». Ajoutant aussi que les conteneurs et les objets OLAP sont définis par des attributs comme le nom, le mode de stockage OLAP et la fonction d'agrégation pour l'objet mesure. Un attribut peut avoir un ensemble de valeurs valables comme MOLAP et ROLAP pour la propriété mode OLAP.

Enfin, on instancie le PDM proposé pour décrire la plateforme cible et qu'on combine, par la suite avec le modèle PIM pour obtenir le modèle OLAP.

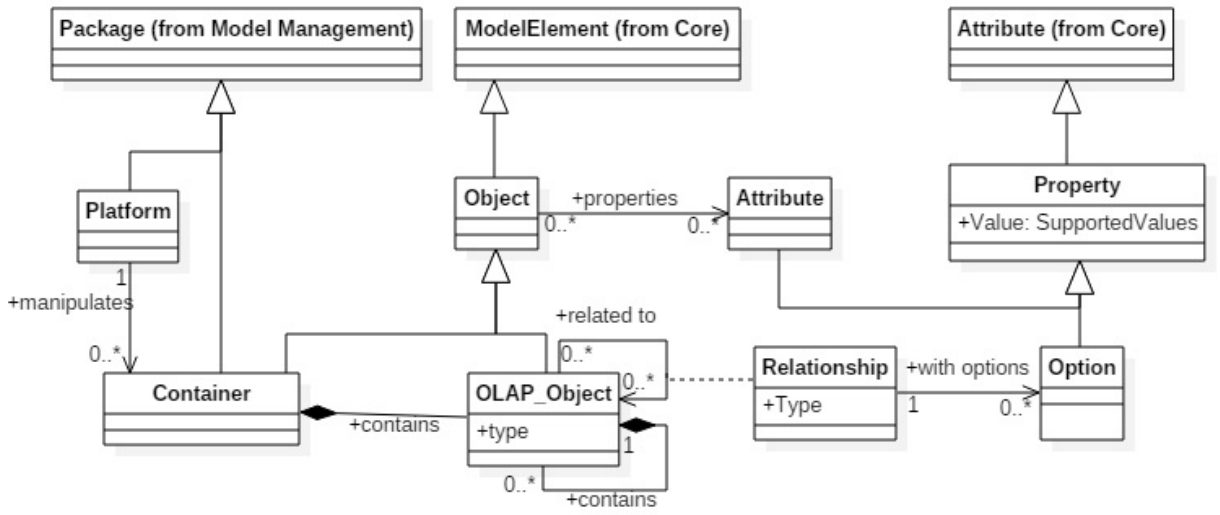


Figure 3.6 Métamodèle PDM pour les plateformes OLAP

3.5. Modèle PSM

Dans notre approche MDA, nous générons deux modèles spécifiques à partir du modèle conceptuel, qui sont le modèle relationnel du DW et le modèle OLAP du cube associé, pour générer par la suite le code d'implémentation correspondant à chaque plateforme. Pour les modéliser, nous avons opté pour les métamodèles du standard OIM comme expliqué dans la section 2.5. Nous décrivons ci-dessous chacun de ces deux métamodèles.

3.5.1. Modèle Relationnel

Le modèle relationnel, tel que proposé par l'OIM (MDC-OIM, 1999) contient principalement un schéma, composé de tables qui héritent de la classe ColumnSet. Cette dernière contient des colonnes et des clés (voir Figure 3.7). Une clé est elle aussi une colonne, pouvant être de type primaire ou étrangère. Le métamodèle OIM comprend d'autres classes qui représentent les contraintes, procédures, vues, index, triggers, types de données etc.

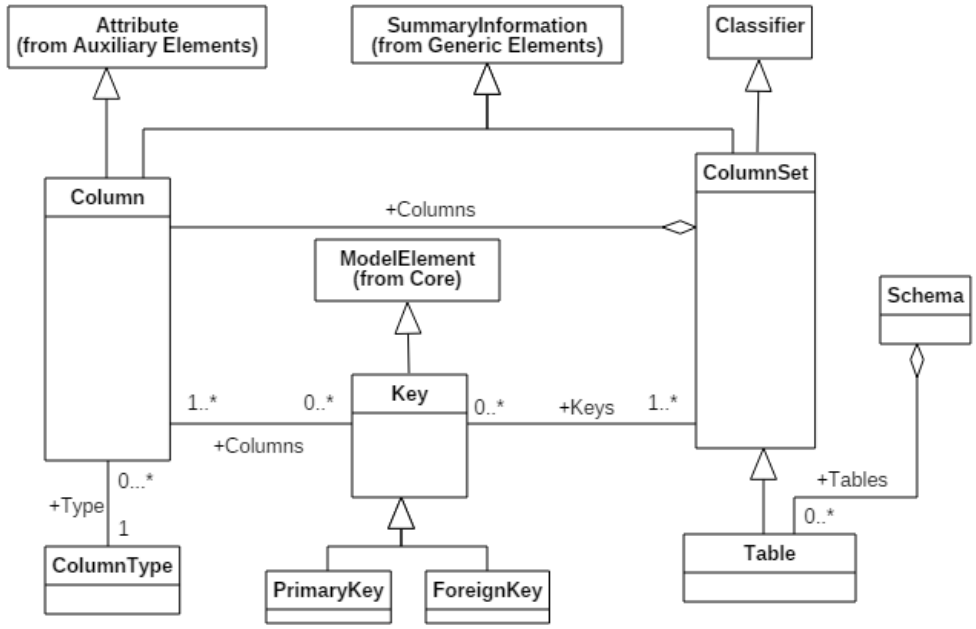


Figure 3.7 Un extrait du métamodèle OIM-Relationnel (MDC-OIM, 1999)

Dans un DW, le modèle relationnel est composé d'une table centrale (table de fait) reliée à des tables de dimension, dans le cas d'un schéma en étoile ou en flocon de neige. Il est, par contre, composé de plusieurs tables de fait partageant des tables de dimension dans le cas de schéma en constellation. La clé primaire de la table de fait est composée des clés primaires de ses tables de dimension, en plus des colonnes relatives aux dimensions dégénérées, si elles existent. La table de fait contient également des colonnes numériques correspondant aux mesures. Enfin, les tables de dimension contiennent en plus de leurs identifiants, des colonnes relatives aux attributs de dimension.

3.5.2. Modèle OLAP

Le métamodèle OLAP proposé par l'OIM décrit le format des objets multidimensionnels (MDC-OIM, 1999). Dans ce schéma, tel que illustré par la Figure 3.8, les cubes et dimensions se rapportent à une base de données OLAP. Cette dernière est, à travers la classe

DeployedOLAPDatabase, physiquement créée au niveau du serveur OLAP, fournisseur physique des métadonnées multidimensionnelles. En plus, les bases de données OLAP contiennent les objets de type connexion, décrivant les chaines de connexion requises.

Un cube peut être persistant « PhysicalCube » ou non persistant « VirtualCube » dérivé d'autres cubes persistants. Un cube physique est composé d'un ensemble de partitions. En fait, une partition est un sous-ensemble d'un cube créé dans le but d'améliorer les performances du cube, faciliter son maintenance et permettre la gestion de l'espace de stockage (MDC-OIM, 1999).

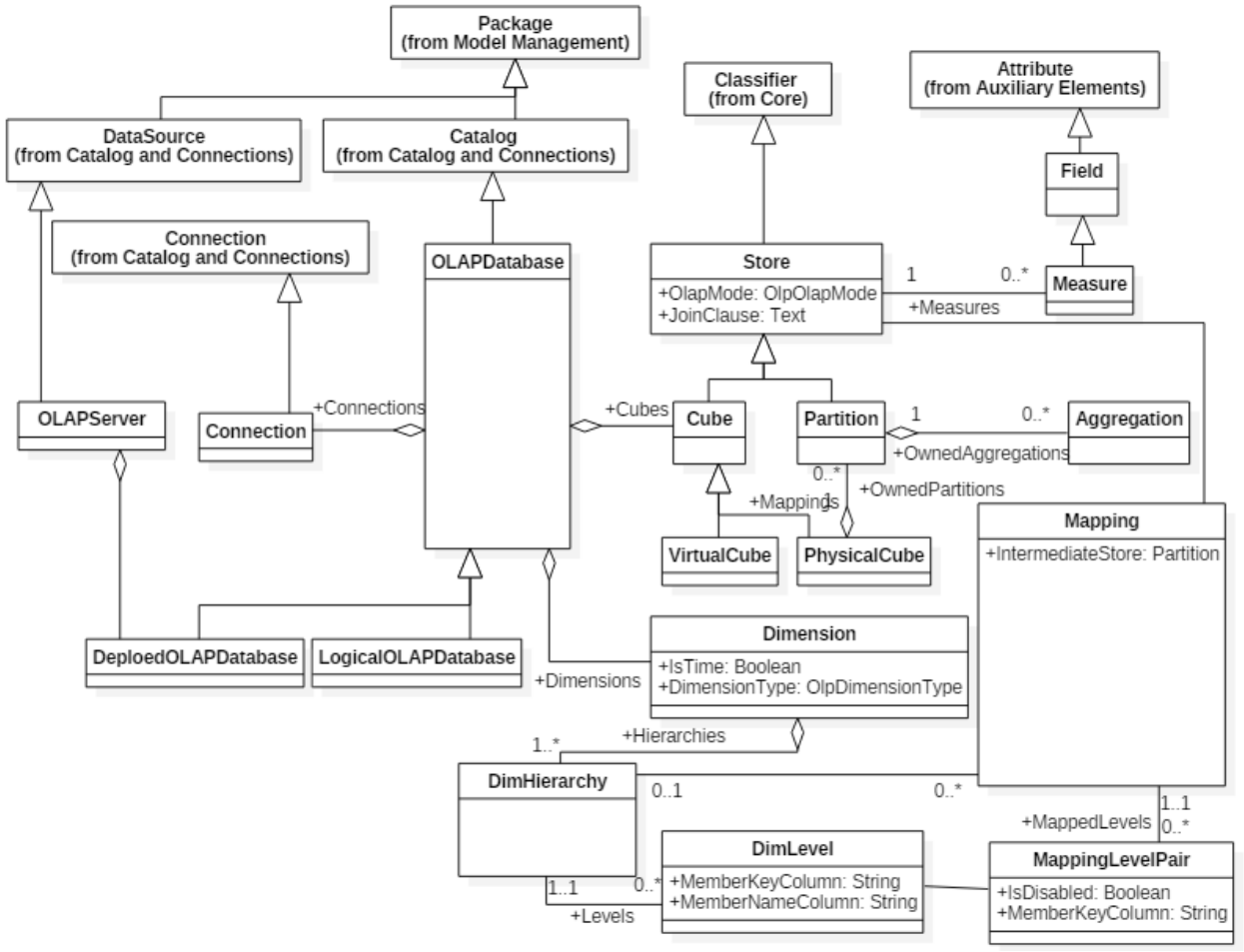


Figure 3.8 Métamodèle OIM-OLAP (MDC-OIM, 1999)

Les cubes et partitions représentent un ensemble de mesures. Un cumul pré-calculé de ces dernières est une agrégation. En outre, les classes cube, partition et agrégation héritent de la classe abstraite Store, au quelle on affecte un mode de stockage, à travers son attribut OlapMode qui prend les valeurs 1 pour le mode hybride (HOLAP), 2 pour relationnel (ROLAP) et 3 pour multidimensionnel (MOLAP).

D'autre part, une dimension est composée de hiérarchies (DimHierarchy) représentant une classification de niveaux (DimLevel). Une dimension est caractérisée par un type pouvant être

régulier, temps, quantitatif ou autre. Enfin, la classe Mapping est utilisée pour relier un objet Store à une hiérarchie de dimension ou à un niveau hiérarchique via la classe MappingLevelPair.

3.6. Règles de transformation

Notre approche MDA a pour objectif d'automatiser la mise en place des systèmes décisionnels. Ainsi, le passage d'une phase du cycle développement du système vers une autre se fait de manière semi-automatique (CIM2PIM), voir automatique (PIM2PSM et PSM2Code). Pour ce faire, nous définissons un ensemble de règles de transformation de type modèle-à-modèle afin de passer d'un niveau vers un autre, et modèle-à-texte pour générer le code d'implémentation. Les deux types de transformation sont implémentés avec le langage de transformation ATL (ATL, 2006). Ci-dessous la liste des règles de transformation principales relatives à chaque niveau.

3.6.1. CIM à PIM

a. Règles de transformation automatiques

Règle 1 : à chaque « DataRequirement » DR, correspond un schéma en étoile et un fait F dans le modèle PIM. Ce fait est composé d'attributs de fait ou mesures $FA= \{fa_1,..,fa_n\}$, correspondant aux éléments OUTDO du modèle, plus précisément aux mesures $M= \{m_1,..,m_n\}$.

DR (M) → F(FA)

Règle 2: à chaque dimension D dans le SGAP, caractérisée par un ensemble d'attributs $DA=\{da_1,..,da_k\}$, correspond une dimension D_C dans le modèle conceptuel, ayant le même nom que D et comme attributs de dimension les attributs de D et ayant une relation de type one-to-many avec le fait F créé précédemment.

D (DA) → D_C(DA)

Règle 3: à chaque dimension D dans le SGAP, n'ayant pas d'attributs, correspond une dimension dégénérée D_D dans le modèle conceptuel précisant son type via la propriété *IsDegenerate*.

D → D_C(IsDegenerate) tel que *IsDegenerate=True*

b. Règles de raffinement manuelles du modèle

Le raffinement du modèle PIM obtenu automatiquement concerne deux principaux traitements, à savoir la réorganisation du modèle et son enrichissement avec des métadonnées suite à la réconciliation avec les sources de données précisées dans le SGAP :

Règle 1 : Regroupement de plusieurs faits en un seul

Tous faits F_1 et F_2 ayant la même dimensionnalité $D = \{d_1, d_2, .., d_m\}$, sont regroupés dans un seul fait *F*, composé de l'ensemble des mesures des deux faits M_1 et M_2 respectivement et ayant la même dimensionnalité que F_1 et F_2.

$F_1(M_1, D) \cup F_2(M_2, D) \rightarrow F(M,D)$ tel que $M = M_1 \cup M_2$

Règle 2 : Définition des hiérarchies

Pour chaque dimension, les attributs de dimension constituant une hiérarchie sont transformés en hiérarchie, tel que chaque attribut A_i est transformé en une base du modèle conceptuel ayant une association de type roll-up-to vers l'attribut A_{i+1} représentant la base du niveau suivant dans la hiérarchie.

$A = \{A_1, .., A_r\} \rightarrow H(B)$ tel que $B = \{B_1, .., B_r\}$ et $B_i = A_i$ où $1 \leq i \leq r$

Règle 3 : Enrichissement avec des métadonnées

Pour chaque attribut de fait ou de dimension, on spécifie son type et sa taille, suite à la réconciliation du modèle avec les sources de données précisées dans le SGAP.

Règle 4 : Traitement de la relation plusieurs-à-plusieurs

Pour toute relation entre fait et dimension de type many-to-many, on précise la table de fait intermédiaire ou fait dégénéré.

Règle 5 : Ajustement des nomenclatures des classes

On renomme chaque fait du modèle selon les mesures qu'il contient et selon la nomenclature convenue, de même pour les mesures, dimensions, et attributs de dimensions.

3.6.2. PIM à Relationnel

Dans toutes les approches de la littérature, le schéma utilisé pour le DW est celui en étoile. Comme vu dans la section 1.3, chacun des deux modèles possibles étoile et flocon de neige possède des avantages et des inconvénients. Le choix d'un modèle reste subjectif et la combinaison des deux est également possible. Pour cela, nous présentons, dans ce qui suit, les règles de transformation pour l'obtention des deux modèles, étoile et flocon de neige, à partir du modèle conceptuel.

Règle 1 : consiste à générer pour chaque fait *F*, une table *T* dans le modèle relationnel. La table générée a pour colonnes $Col_F = \{col_{F1}, .., col_{Fx}\}$, les attributs de fait (mesures) $FA = \{fa_1, .., fa_n\}$ ainsi que les dimensions dégénérées $DD = \{dd_1, .., dd_p\}$ de *F*, avec leurs même noms et types (convertis en types relationnels correspondants) en plus des clés de ses dimensions

$D = \{d_1, .., d_m\}$ de type un-à-plusieurs. En outre, la table générée a comme clé primaire *PK*, l'union des clés de ses dimensions D de type un-à-plusieurs, en plus de ses dimensions dégénérées *DD*. Enfin, la table de fait T a comme clés étrangères $FK = \{fk_1, .., fk_m\}$, les clés de ses dimensions de type un-à-plusieurs.

$F\,(FA, D, DD) \rightarrow T\,(Col_F, PK, FK)$

Règle 2 : dans le cas d'un schéma en étoile, chaque dimension est transformée en une table relationnelle T ayant comme colonnes $Col_D = \{col_{D1}, .., col_{Dy}\}$, les attributs de dimension de ses bases $DA = \{da_1, .., da_k\}$, leurs descripteurs $Des = \{Des_1, .., Des_l\}$ et $OID = \{OID_1, .., OID_l\}$ à l'exception du OID de la base terminale.

Comme clé primaire PK, on crée une nouvelle clé composée de la concaténation du préfix « ID_ » et le nom de la dimension (voir Figure 3.9).

$D\,(DA, Des, OID) \rightarrow T\,(Col_D, PK)$

Par contre, dans le cas d'un schéma en flocon de neige, chaque base B ayant un *OID* est transformée en une table. La base terminale, base directement liée à la dimension, est alors mappée à la table de dimension reliée directement à la table de fait. La table générée a comme colonne l'*OID* de la base terminale, son descripteur et ses attributs de dimension et comme clé primaire *PK*, on crée une nouvelle clé (clé de substitution) composée de la concaténation du préfix « ID_ » et le nom de la dimension et comme clé étrangère l'*OID* de la base directement reliée à cette dernière, et de même pour les autres bases.

$B\,(OID, Des, DA) \rightarrow T\,(Col_b, PK, FK)$

Règle 3 : chaque fait dégénéré est transformé en une table d'association T reliant les tables de dimension associées. T a comme colonnes Col_{DF} les attributs de fait associés FA_{DF} ainsi que les clés des ses dimensions de type un-à-plusieurs D_{DF}, y compris la clé de la dimension concernée par la relation plusieurs-à-plusieurs.

$DF\,(FA_{DF}, D_{DF}) \rightarrow T\,(Col_{DF}, PK, FK)$

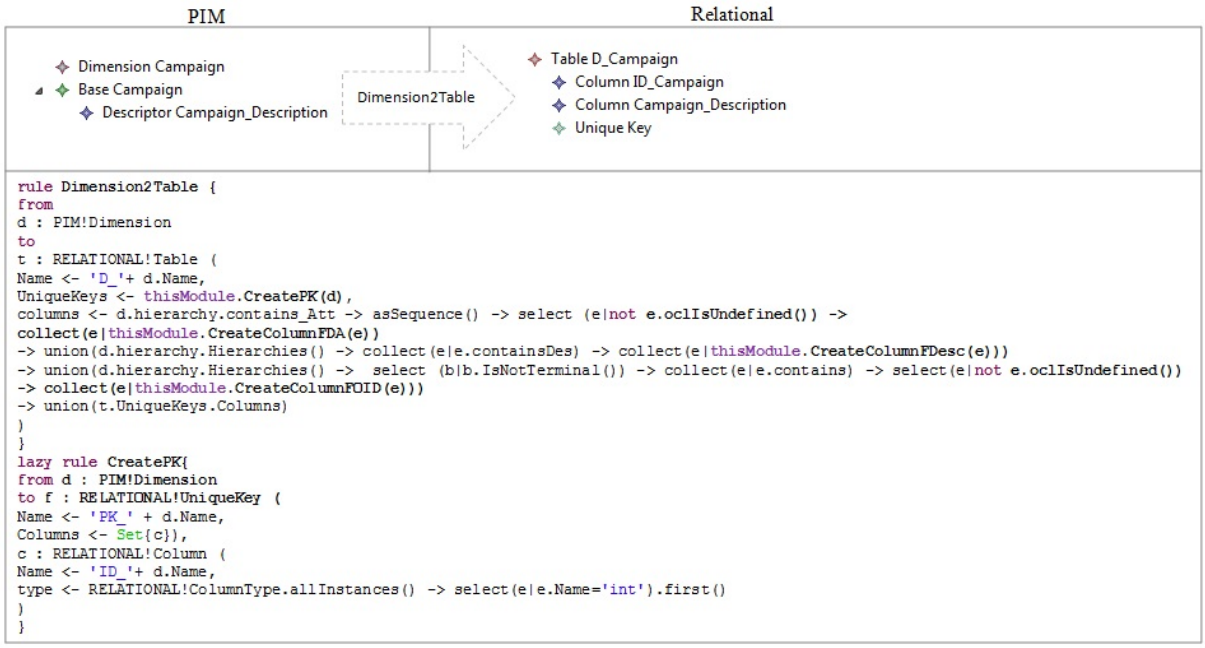


Figure 3.9 Règle de transformation de dimension dans le PIM vers table

3.6.3. PIM à OLAP

La transformation du modèle PIM en modèle OLAP, doit prendre en considération les limitations de la plateforme, en particulier la prise en charge du type de relation plusieurs-à-plusieurs et les modes de stockage supportés. Ainsi, pour garantir une transformation automatique valable pour n'importe quel outil OLAP, on utilise un modèle PDM qui décrit la plateforme cible. Les règles de transformation principales pour générer le modèle OLAP à partir du modèle conceptuel sont les suivantes :

Règle 1 : chaque fait F dans le modèle conceptuel est transformé en un cube C dans le modèle OLAP ayant le même nom que F et un mode de stockage par défaut, MOLAP ou ROLAP, selon le modèle PDM. La transformation inclut aussi la génération d'une partition par défaut P. De plus, les attributs de fait $FA = \{fa_1, .., fa_n\}$ sont transformés en mesures $M = \{m_1, .., m_n\}$ dans le modèle OLAP.

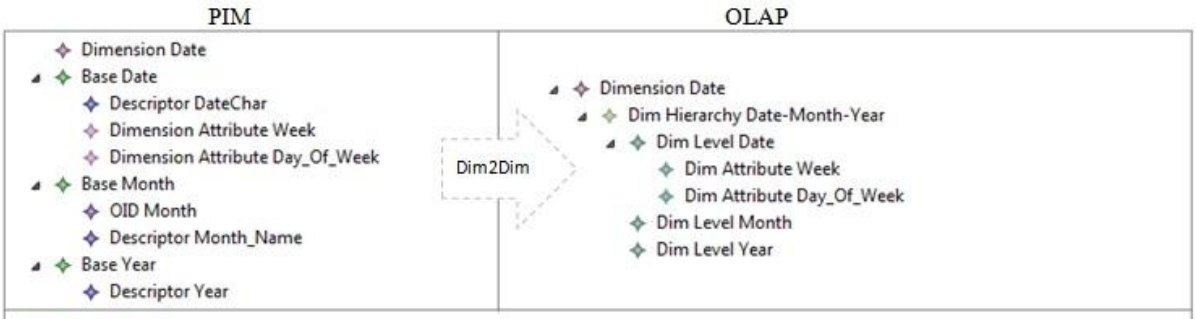
$F\,(FA) \rightarrow C\,(P, M)$

Règle 2 : chaque dimension D est transformée en une dimension OLAP D_o (Figure 3.10). La transformation implique aussi la génération des hiérarchies associées H ainsi que les niveaux $L = \{l_1, .., l_r\}$ de chaque hiérarchie. Pour ce faire, on utilise une fonction qui construit les hiérarchies en parcourant les bases $B = \{b_1, .., b_r\}$ de la dimension. Comme convention, le nom N_h de chaque hiérarchie générée est la concaténation des noms de ses niveaux triés et séparés par des '-'. Le

descripteur *Des*, l'*OID* et les attributs de dimensions $DA=\{da_1,..,da_k\}$ de chaque base sont transformés en MemberNameColumn *MK*, MemberKeyColumn *MN* et attributs de dimension $DA_o= \{da_{o1},..,da_{ok}\}$, respectivement. Ajoutant aussi que la classe relationship *A* reliant la dimension *D* et le fait *F* (ou fait dégénéré) est transformée en :

- Dans le cas d'une relation un-à-plusieurs, on génère un objet M_p qui relie la partition résultante *P* à la dimension *D*.
- Dans le cas d'une relation plusieurs-à-plusieurs, l'objet de mapping généré M_p indique aussi l'objet Store intermédiaire reliant le fait courant à sa dimension.

D (B (OID, Des, DA), A) $\rightarrow D_o$ *(H (L), MK, MN,* DA_o*, Mp)*




```
rule Dim2Dim {
from
d : PIM!Dimension (not d.isTreeElement())
to
t : OLAP!Dimension (
Name <- d.Name,
Hierarchies  <- h
),
h : OLAP!DimHierarchy(
Name  <- d.hierarchy.Name.Hierarchies()
-> flatten() -> iterate(n ; nm : String=''|
if (n=OclUndefined) then
nm
else
if nm='' then n else nm+'-'+n endif
endif
),
ordred <- d.hierarchy.Name.Hierarchies()-> asSequence()-> collect(e|thisModule.CreateLevel(e))
)
};
helper context String def : Hierarchies() : OrderedSet(String) =
let Bases : OrderedSet(PIM!Base) =
PIM!Base.allInstances() ->asSequence() in
Bases  -> select(c|c.Name=self) -> flatten()-> first().rolls_up_to ->asSequence() ->
iterate(r; b :OrderedSet(PIM!Base)= OrderedSet{self} |
if r.rolls_up_to=OclUndefined or r.rolls_up_to.isEmpty() then
b.append(r.Name)
else
b.union(r.Name.Hierarchies())
endif
);
```

Figure 3.10 Règle de transformation d'une dimension du PIM vers une dimension dans le PSM

Règle 3 : la transformation du fait dégénéré dépend de la prise en charge ou non de la relation plusieurs-à-plusieurs par la plateforme cible. Si ce type de relation est supporté, le fait dégénéré *DF* reliant un fait *F* à une dimension *D* est transformé en une partition P_{df}. Les attributs de fait FA_{df} de *DF,* sont transformés en mesures M_{df} de P_{df}.

$DF(\mathrm{FA_{df}}) \rightarrow \mathrm{P_{df}}(\mathrm{M_{df}})$

Cependant, dans le cas où la relation plusieurs-à-plusieurs n'est pas supportée, et pour éviter de perdre des informations du modèle conceptuel, le fait dégénéré DF est transformé en un cube séparé ayant comme dimensions, celles associées à DF (comme décrit dans la règle 1).

$DF(\mathrm{FA_{df}}) \rightarrow \mathrm{C_{df}}\ (\mathrm{P_{df}}\ , \mathrm{M_{df}})$

Règle 4: le package Pkg du modèle conceptuel est transformé en une base de données OLAP DB_o composée des cubes et dimensions créés par les règles précédentes.

$Pkg \rightarrow DB_o(C, D)$

3.6.4. PSM à Code

La création d'un cube OLAP reste une tâche étroitement liée à la plateforme utilisée et qui nécessite une bonne connaissance de l'outil et de ses assistants, représentant une boite noire pour les développeurs. Pour cette raison, notre approche vise à permettre la création automatique des cubes OLAP via du code.

Ainsi, nous avons commencé par mener une étude comparative des plateformes BI les plus utilisées. Nous avons déduit alors que le code d'implémentation des cubes varie d'une plateforme à l'autre. Le tableau 1.1 (Chapitre 1, section 1.5) liste certaines des spécificités et extensions des cubes pour certains éditeurs OLAP.

Comme exemple d'implémentation, nous traitons dans ce qui suit la génération du code XMLA (XML for Analysis) relatif au Framework SSAS (SQL Server Analysis Services). En fait, le XMLA est un protocole XML basé sur le SOAP (Simple Object Access Protocol). C'est un standard conçu pour accéder aux données à travers des systèmes multidimensionnels. Le XMLA a été proposé par Microsoft en avril 2000 et il est maintenant supporté par plus de 25 éditeurs comme SAS, Hyperion, Cognos, et SAP Net Weaver BW. Le XMLA prend en charge deux méthodes principales pour accéder aux données; execute et discover. Microsoft a étendu ce langage en tant que ASSL (Analysis Services Scripting Language) pour permettre de manipuler et contrôler des objets multidimensionnels via des commandes comme create, delete et process.

D'une autre part, pour créer le cube OLAP, le DW relationnel doit être déjà créé. Le code SQL pour la création des tables du DW est généré depuis le modèle relationnel, il correspond aux commandes « Create table » et « Alter table » pour la création des tables et des associations. Avant chaque transformation, la règle vérifie si la table est déjà créée. En effet, plusieurs dimensions peuvent utiliser la même table, comme par exemple, le cas de deux dimensions date "Date vente" et "Date livraison", les deux utilisent la table Date. Enfin, la table créée est mise à jour avec les nouvelles associations.

Par la suite, pour générer le code d'implémentation du cube, les deux modèles résultants de la phase de conception, OLAP et relationnel, sont utilisés (voir Figure 3.11). En fait, le modèle OLAP sert à définir les structures multidimensionnelles, tandis que les propriétés de ces structures comme la table et la colonne relationnelles associées, son type et sa taille, sont déduits du modèle relationnel.

La correspondance entre les deux modèles, OLAP et relationnel, se fait via les noms des objets. Le premier objet décrit par le fichier XMLA du cube est la base de données OLAP définie par un nom et un identifiant. On décrit ensuite les dimensions contenues par cette base de données ainsi que leurs attributs, hiérarchies et niveaux hiérarchiques. En outre, le fichier définit pour chaque attribut, la table relationnelle associée ainsi que la colonne correspondante, son type et sa taille.

Le fichier décrit ensuite le cube et liste encore une fois toutes les dimensions associées à ce dernier, leurs attributs et hiérarchies.

Outre cela, le fichier décrit pour chaque groupe de mesures correspondant à une table de fait, la table relationnelle associée ainsi que la colonne et le type de chacune de ses mesures. Dans cette section, le fichier XMLA liste également, les dimensions associées à chaque table de fait, ainsi que le type de relation reliant ces deux et qui peut être régulier (si la dimension est directement reliée à la table de fait), référencé (si la dimension est reliée à une dimension intermédiaire, qui à son tour, est reliée à la table de fait), fait (dans le cas de dimension dégénérée) ou plusieurs-à-plusieurs (si la dimension est reliée à une table de fait intermédiaire qui est reliée à une dimension intermédiaire et qui, à son tour, est reliée à la table de fait).

On décrit par la suite les partitions. Chaque partition est associée à une table de fait et possède un mode de stockage (MOLAP, ROLAP ou HOLAP).

En outre, le fichier XMLA définit la source de données caractérisée par un nom et une chaine de connexion relative au DW relationnel, extraite depuis la classe « connection » du modèle OLAP.

Enfin, le fichier décrit la vue de source de données. Cette dernière est une abstraction de la source de données relationnelle et qui contient les tables utilisées par le cube. Elle décrit les tables de faits et de dimensions, leurs colonnes et leurs clés primaires, ainsi que les clés étrangères et les relations entre ces tables.

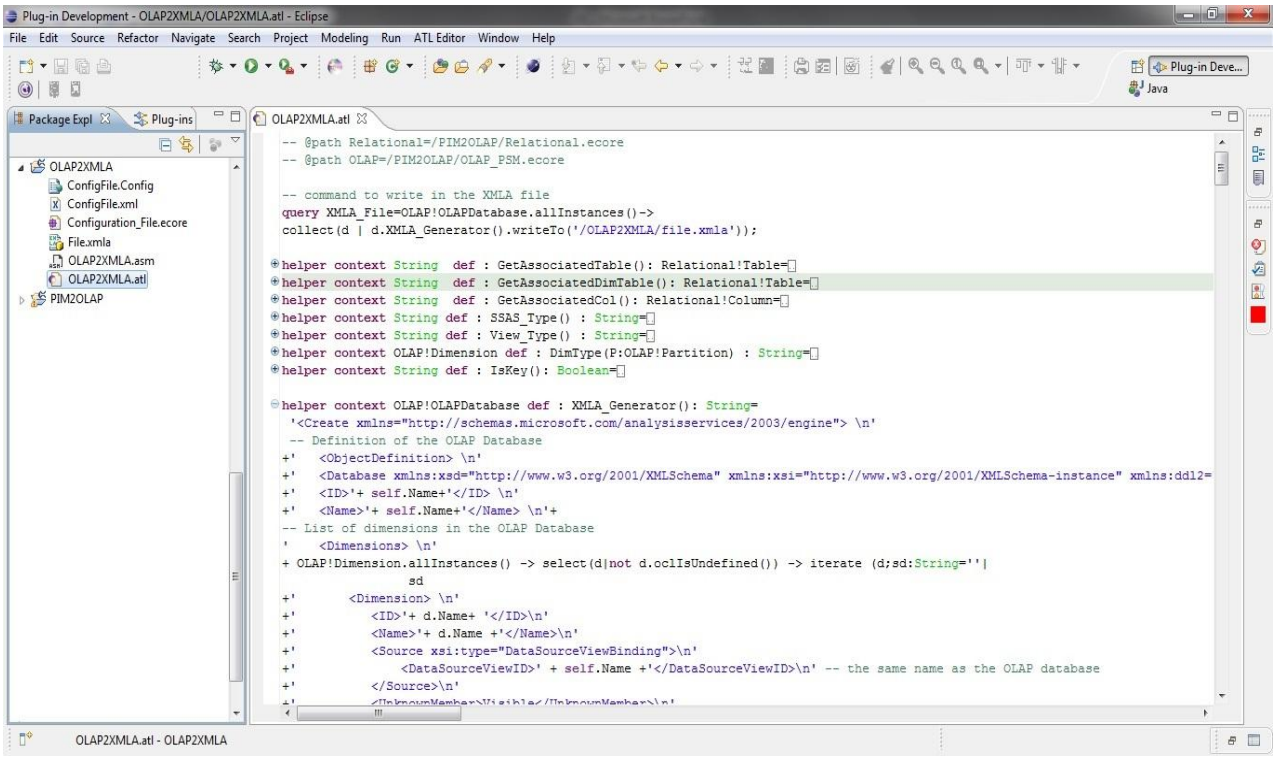


Figure 3.11 Aperçu sur le fichier de transformation ATL pour la génération du code XMLA

3.7. Collecte automatique des métadonnées

Dans une architecture MDA, les modèles générés (CIM, PIM, PSM) constituent la documentation de haut niveau du nouveau système décisionnel (Kleppe, 2003). Toutefois, cette documentation peut devenir inexploitable après l'expansion du système, vu le grand nombre de modèles qui seront alors créés. En outre, bien que la majorité des SGBD et éditeurs OLAP fournissent des dictionnaires de données décrivant les métadonnées techniques du DW et cubes OLAP respectivement, le lien entre les entités des deux plateformes n'est pas disponibles. Par exemple, il est difficile de connaître la liste des cubes où une table particulière est utilisée comme dimension pour mesurer l'impact d'une modification au niveau de la structure de cette table.

Pour cela, notre approche MDA intègre la collecte et le répertoriage des métadonnées disponibles tout au long du processus de développement du projet, dans un entrepôt de métadonnées (voir Figure 3.12). Ce dernier peut être une base de données relationnelle ou tout autre type de fichiers. Il sert d'une part à documenter le système, mais aussi à tracer la lignée ou la provenance des données (data lineage), ce qui constitue des éléments indispensables pour la gouvernance des données et la gestion des changements.

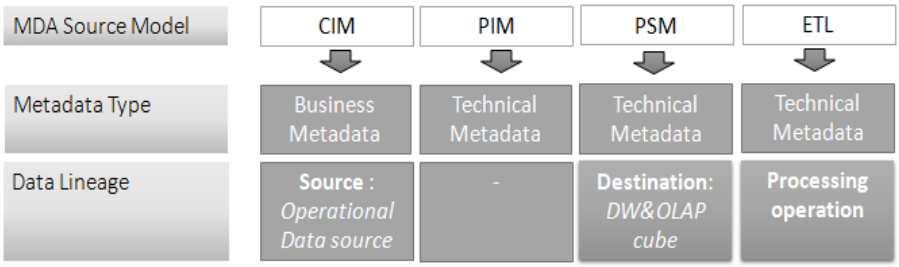


Figure 3.12 Notre approche de collecte des métadonnées et lignée des données

3.7.1. Modélisation des métadonnées

En général, les métadonnées doivent fournir des réponses aux cinq questions suivantes (Tannenbaum, 2002) : qu'est ce qu'on a ? Qu'est ce qu'il signifie ? Où est-il? Comment a-t-il parvenu ici et comment on peut l'avoir ?

Pour fournir des métadonnées qui répondent à ces questions, nous classons d'abord les métadonnées en deux catégories : business et technique, comme proposé par (Kimball, 2013) et (Ponniah, 2004). Les métadonnées business concernent, principalement, la description métier de chaque donnée, ainsi que la description des buts, objectifs et leurs responsables. Alors que les métadonnées techniques fournissent la description des sources de données nécessaires pour charger ou livrer les données, comme par exemple l'URL, le fournisseur, le type (relationnel, OLAP etc.) et le rôle de la source de données dans la lignée des données, en plus de la description des tables, colonnes, cubes, dimensions, hiérarchies etc.

Les métadonnées techniques incluent également les métadonnées de traitement, notamment, la description du processus de chargement des données, le calendrier des mises à jour, ainsi que la description des différentes opérations qu'il inclut (fusionner, convertir, agréger, charger etc.).

En outre, nous inscrivons les règles techniques et opérationnelles comme métadonnées techniques ou business respectivement.

D'autre part, nous regroupons les métadonnées par MetadatSet (MDS) afin de permettre de regrouper les métadonnées relatives au même sujet dans le même groupe, comme par exemple les métadonnées relatives à une source de données (URL, fournisseur, type, serveur etc.). Ajoutant également, qu'un metadataSet peut dépendre d'un autre. Enfin, nous distinguons dans les metadataSets entre ceux descriptifs et ceux de traitement, comme le montre la Figure 3.13.

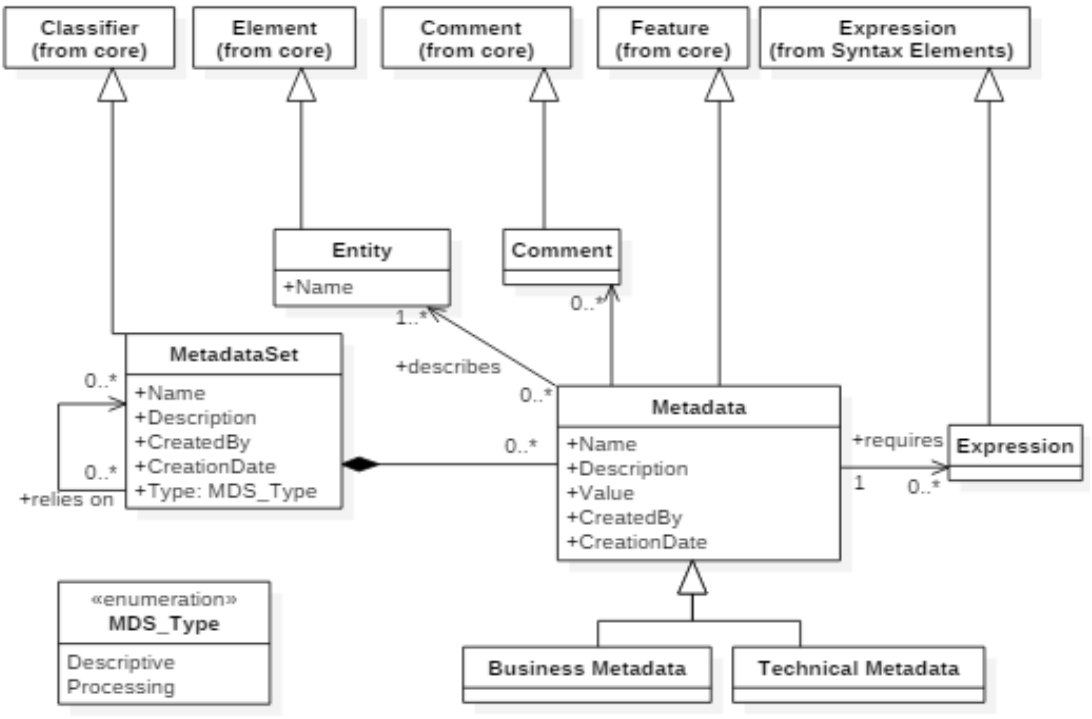


Figure 3.13 Modèle des métadonnées des DSS

3.7.2. Les règles de transformation pour la collecte des métadonnées

Notre approche consiste à collecter et répertorier les métadonnées disponibles dans les niveaux MDA. Ainsi, toute description d'une entité E dans un modèle source (CIM, PIM, PSM) est une métadonnée *m(Nom,Valeur,E)* définie par son nom, sa valeur et l'entité qu'elle décrit et faisant partie d'un metadataSet.

a. Métadonnées issues du CIM

Notre modèle CIM conçu avec nos profils GoalCases et SGAP, fournit toutes les métadonnées métier concernant les buts et les objectifs du projet. Il fournit également la description métier de chaque entité ainsi que sa source de données. On définit également au niveau du CIM, le responsable du but et les règles métier à respecter. Ainsi, à partir du modèle GoalCases, on extrait pour chaque objet « QualifyBy » Q, la liste de ses attributs A qu'on répertorie comme des métadonnées de type business et qu'on regroupe dans un metadataSet de type descriptif. Nous pouvons résumer cette transformation par :

$Q(A) \rightarrow E(M)$ où M est un metadataSet décrivant l'entité E correspondant à Q, telle que chaque métadonnée m_i dans M correspond à un attribut de Q.

Par exemple, en supposant que Q a été déclaré dans le GoalCases comme étant un KPI, nommé « Qualité de service » dont l'unité est le pourcentage, on aura alors parmi les métadonnées extraites :

$m_1("Type", KPI, Qualité\ de\ service)$, $m_2("Unit", \%, Qualité\ de\ service)$

On répertorie à ce niveau également, le but du projet, son responsable ainsi que le descriptif de l'objectif associé à Q.

Outre cela, d'autres entités sont décrites dans le modèle SGAP, en particulier celles relatives aux dimensions. On décrit également à ce niveau les sources de données « Repository » requises pour l'extraction de ces entités. On extrait alors les métadonnées relatives à ces entités en plus des métadonnées décrivant les sources de données.

Ainsi, nous ajoutons, par exemple pour la « Qualité de service », la métadonnée :

$m_3("Source", DB\_Sales, Qualité\ de\ service)$, en supposant que DB_Sales est la source de données de la « Qualité de service ».

b. Métadonnées issues du PIM

Dans notre architecture OMDA, le modèle PIM fournit une représentation multidimensionnelle indépendamment de la technologie. En termes de métadonnées, le modèle PIM fournit des métadonnées techniques additionnelles pour les entités décrites dans le CIM. Ainsi, pour chaque attribut de fait, la règle de dérivation ainsi que la fonction d'agrégation, sont ajoutées comme métadonnées techniques regroupées dans un metadataSet. On ajoutera, par exemple, une métadonnée de type technique, décrivant la règle de dérivation de la mesure « Qualité de service », telle que $m_4("Formula", NbVentes/NbRetours, Qualité\ de\ service)$

De même pour les attributs de dimension, on ajoute les métadonnées techniques décrivant leurs règles de dérivation.

c. Métadonnées issues du PSM

La couche PSM fournit les métadonnées techniques relatives aux plateformes cibles, le DW relationnel et cube OLAP. On obtient alors, à partir de ce niveau, toutes les métadonnées techniques qui décrivent les entités créées auparavant. Ainsi, on extrait pour chaque entité ses métadonnées techniques: table, colonne, taille, type, dimension, hiérarchie, niveau hiérarchique etc. On ajoute également deux groupes de métadonnées de type « Processing », décrivant le DW et le cube OLAP en indiquant le rôle de ces deux dans la lignée de données. Le cube est alors décrit comme étant une plateforme cible « Delivery system», alors que le DW est à la fois une cible mais aussi une source vis-à-vis du cube.

Comme exemple de métadonnées de la mesure « Qualité de service », on aura :

m_5(“Table”, F_Sales, Qualité de service),m_6(“Column”, $NbVentes$, Qualité de service),

m_7(“Column”, $NbRetours$, Qualité de service), m_8 (“Cube”, $Sales$, Qualité de service)

d. Modèle ETL

Dans cette section, nous évoquons l'intégration du modèle ETL dans l'approche de génération des métadonnées. En fait, un processus ETL consiste, en général, à charger des données depuis une source vers une destination suivant un ensemble d'opérations de transformation. Pour modéliser ce processus, de nombreuses méthodes existent. En effet, des langages standards comme l'UML, proposé par (Munoz et al, 2008 ; Trujillo et al, 2003), DSL (Domain Specific Language) proposé par (Petrović et al, 2017 ; Vassiliadis et al, 2005) et BPMN (El Akkaoui et al, 2012) ont été étendus de façon à supporter les concepts des ETLs. Dans notre approche nous utilisons le diagramme d'activité UML (Munoz et al, 2008) qui, grâce à sa flexibilité, permet de modéliser tout type d'opérations.

En fait, le modèle d'un ETL fournit les métadonnées de traitement et une partie considérable de la lignée des données. Ainsi, nous extrayons à partir de l'ETL, la description de chaque opération de l'ETL, définie par un nom, type (fusionner, charger, filtrer, convertir, agréger, etc.), ordre et paramètres et que nous regroupons dans un même metadataSet. Dans l'exemple relatif à la « Qualité de service », on pourrait obtenir les métadonnées suivantes :

m_9 $(\text{“}OP\ Type\text{”}, Union, Qualité\ de\ service)$, m_{10} $(\text{“}OP\ Ordre\text{”}, 2, Qualité\ de\ service)$

De plus, les métadonnées décrivant la fréquence d'exécution de l'ETL sont également extraites.

La Figure 3.15 illustre un exemple de transformation ETL vers métadonnées :

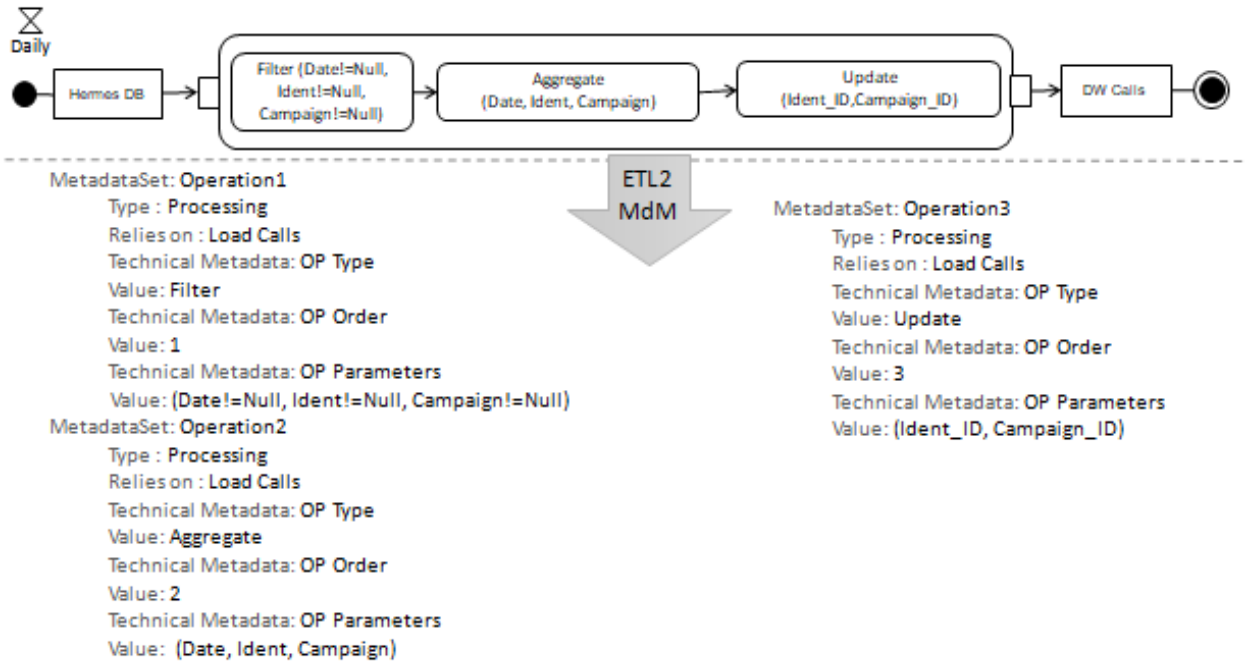


Figure 3.14 Exemple de transformation ETL vers Métadonnées

3.8. Implémentation et étude de cas

Pour implémenter notre démarche MDA pour la mise en place des systèmes décisionnels, nous avons utilisé la plateforme Eclipse. Ainsi, nous avons d'abord créé les métamodèles GoalCases et SGAP en utilisant le Framework GMF (Graphical Modeling Framework) pour la représentation graphique de nos profils, ainsi que le Framework EMF (Eclipse Modeling Framework) pour la création des profils PIM, PDM, OLAP et relationnel. Ce dernier est un framework de modélisation et de manipulation des modèles (fichiers .xmi) et métamodèles (fichiers .ecore), et qui offre également plusieurs outils de génération de code java à partir des modèles. Nous avons, par la suite, implémenté les règles de transformation modèle à modèle et modèle à texte avec le langage ATL. Ainsi, pour chaque passage d'un niveau vers un autre, correspond un fichier de transformation, prenant comme entrée le modèle du niveau en cours et comme sortie le modèle du niveau suivant, conformément aux métamodèles déjà créés.

Dans ce qui suit, nous décrivons l'application de notre démarche MDA sur une étude de cas réel relatif à un centre d'appel souhaitant disposer d'outils lui permettant d'analyser ses prestations, afin de les améliorer et d'identifier les points de faiblesse. La source de données utilisée pour notre étude de cas est le CRM Hermes, le premier CRM de gestion des contacts utilisé dans le monde.

Ainsi, suivant notre démarche orientée but, la première étape est de modéliser les exigences de haut niveau des décideurs, exprimées sous forme de buts stratégiques qu'ils souhaitent atteindre, et ce via notre profil GoalCases. En effet, le but principal dans l'étude de cas traitée, tel que défini par la direction générale, l'acteur « DecisionMaker », est d'améliorer la prestation des campagnes d'émission d'appels, appelées aussi campagnes sortantes. L'achèvement de ce but peut être influencé par la forte concurrence externe, que nous représentons par un « ExternalInfluencer ».

Afin de simplifier l'analyse du but stratégique, nous l'avons décomposé en deux sous buts plus clairs et précis à savoir « Améliorer la qualité de service » et « Augmenter la joignabilité ». Par la suite, pour chaque sous buts, nous définissons précisément les objectifs quantitatifs permettant de réaliser le but associé. Ainsi, pour réaliser le sous but « Améliorer la qualité de service », deux objectifs doivent être réalisés, sous la responsabilité du « Manager produits ». Le premier étant d'« Augmenter la productivité des télé-conseillers », est mesuré par la durée de conversation des appels passés par les télé-conseillers « CallDuration ». Par la suite, à base de cette mesure, on définit les seuils déterminant les trois états du but. Le deuxième objectif est « Augmenter la QS des campagnes sortantes ». Pour le mesurer, on utilise la QS (Qualité de

service), un indicateur de performance clé et qui à son tour détermine les trois états de l'objectif associé. D'une autre part, le sous but « Augmenter la joignabilité » peut être accompli à travers l'achèvement de l'objectif « Améliorer le mécanisme de détection automatique des injoignables » qui est sous la responsabilité du DSI (Directeur du Système d'Information). Ce dernier est mesuré par le ratio de la durée d'attente des appels par rapport à la durée total d'appel. Ainsi, nous aurons obtenu le modèle GoalCases de notre étude de cas, comme illustré par la Figure 3.15.

La deuxième phase est alors de détailler les buts et objectifs identifiés dans le GoalCases. Pour ce faire, nous utilisons notre profil SGAP. Nous commençons alors par insérer un pool A correspondant au but stratégique « Améliorer la prestation des campagnes sortantes » et dans lequel nous ajoutons un lane 1 associé au DecisionMaker « Direction générale ». Nous plaçons alors le but stratégique dans ce lane 1.

Nous utilisons également un autre pool B pour placer le Influencer « Concurrence » et que nous relions au pool A à travers un flux de séquence de type « negative impact ». Nous plaçons ensuite les sous buts, tels que définis dans le GoalCases dans le pool A et que nous relions via un branchement parallèle indiquant que le but stratégique n'est satisfait, que si les deux sous buts le sont.

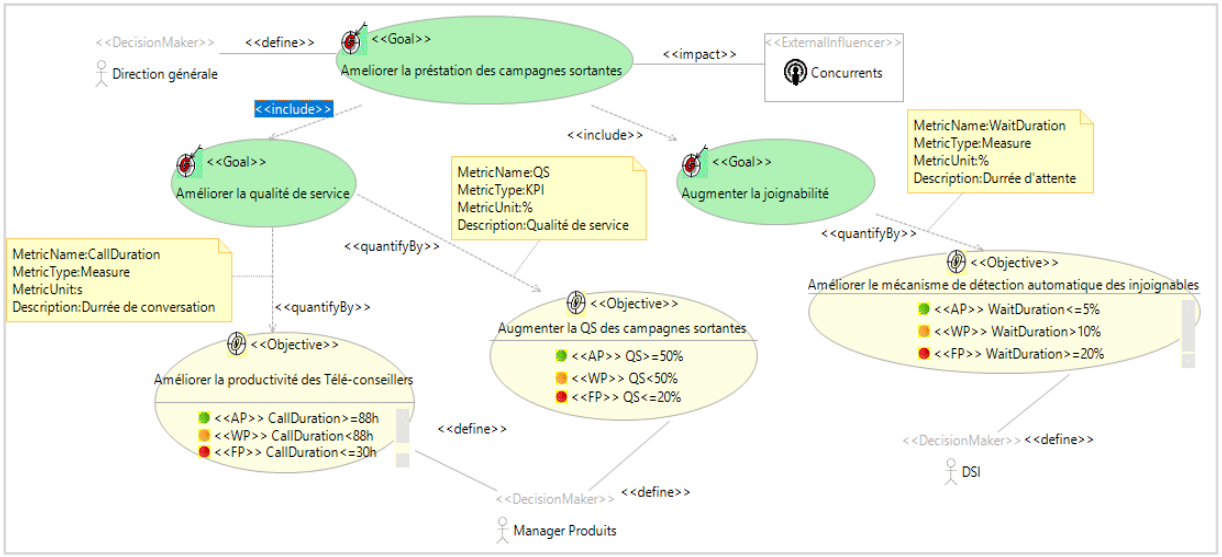


Figure 3.15 Modèle GoalCases de notre étude de cas

L'étape suivante est d'intégrer les objectifs. Pour cela, nous utilisons des lanes que nous incorporons dans le pool A, chaque lane correspondant à un acteur du modèle GoalCases. A savoir le « Manager produits » et la « DSI ». Par la suite, nous plaçons dans chaque lane les objectifs associés à son acteur. Ainsi, dans le lane 2 de l'acteur « Manager produits », nous

plaçons les objectifs « Augmenter la QS des campagnes sortantes » et « Améliorer la productivité des Télé-conseillers », alors que le dernier objectif « Améliorer le mécanisme de détection automatique des injoignables » est placé dans le lane 3 associé à l'acteur « DSI ». Enfin, nous relions les objectifs à leur but via un branchement inclusif, précisant que tous ces objectifs doivent être évalués.

Notant que tous le travail de modélisation fait jusqu'à cette étape au niveau du SGAP, n'est qu'une projection directe du modèle GoalCases vers le SGAP. L'étape suivante est alors de déduire les informations requises à l'accomplissement de chaque objectif. Ainsi, l'objectif « Augmenter la QS des campagnes sortantes» requiert d'« Analyser la QS des campagnes sortantes », alors que l'objectif « Améliorer la productivité des Télé-conseillers **»** requiert d'« Analyser les appels passés par télé-conseiller » et enfin l'objectif « Améliorer le mécanisme de détection automatique des injoignables » requiert d'« Analyser la durée d'attente des appels ». Les informations déduites sont alors reliées à leurs objectifs via un flux de séquence de type « consult ».

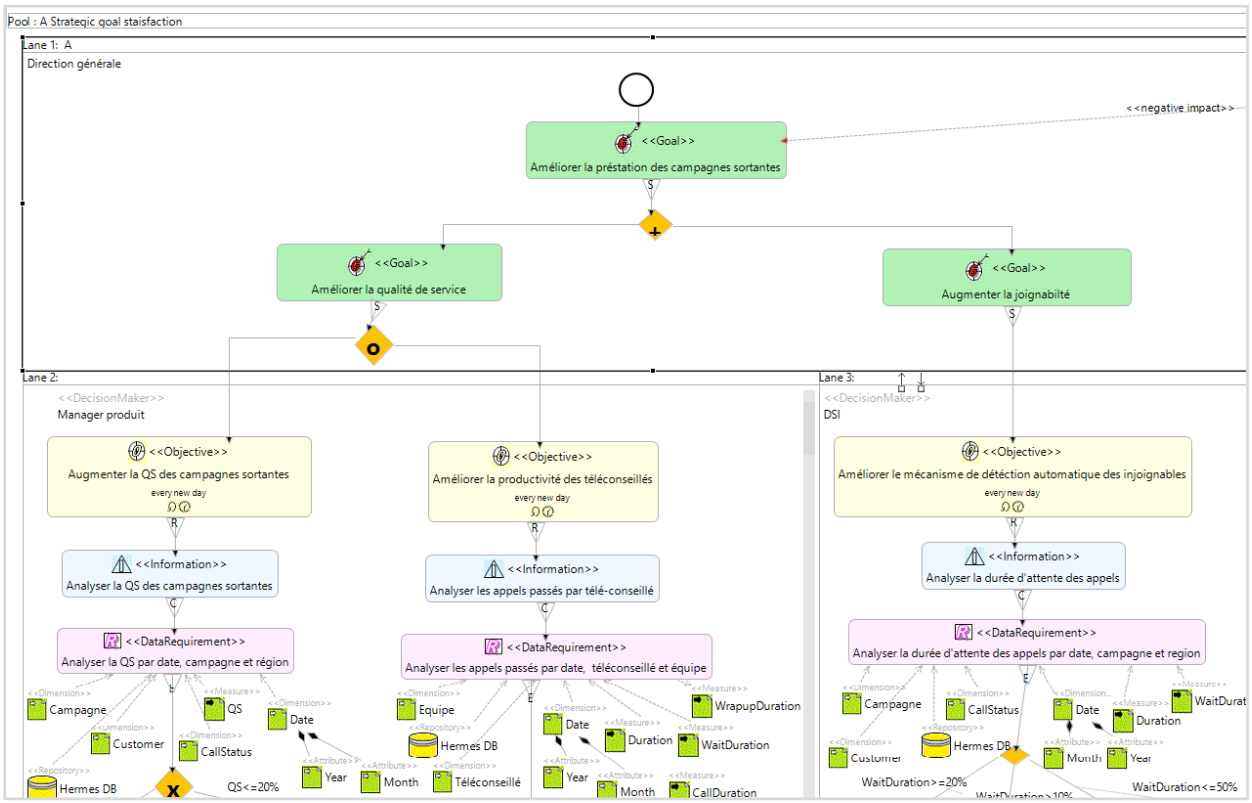


Figure 3.16 Modèle SGAP de notre étude de cas

La dernière étape dans la mise en place du modèle SGAP est de définir les données requises ou « DataRequirement » à chaque information. Pour l'information « Analyser la QS des campagnes sortantes », les données requises sont « Analyser la QS par date, campagne et région » que nous

relions à l'information via un flux de séquence de type « require ». Enfin, nous détaillons les éléments du DataRequirement en distinguant les dimensions, attributs, mesures ou KPI et sources de données. Il s'agit alors d'analyser la durée d'appel de conversation « CallDuration » par rapport à la durée globale « Duration », et ce par date, campagne, client et statut d'appels. Toutes ces données seront extraites de la base de données du CRM Hermes.

Nous intégrons enfin l'évaluation de l'objectif par rapport aux métriques définies. On relie alors les trois états <<AchievementPoint>>, <<WarningPoint>> et <<FaillurePoint>> par un branchement exclusif en indiquant sur chaque branche la condition associée et que nous relions au DataRequirement par un flux de type « evaluate ». En continuant ainsi pour les autres informations, nous recensons les dimensions suivantes et leurs attributs : date, campagne, télé-conseiller, groupe de télé-conseillers, client, statut d'appels et type d'appels et les mesures suivantes : durée de conversation, durée d'attente, durée de traitement et durée globale.

Nous obtenons alors le modèle SGAP comme le décrit la Figure 3.16.

Ainsi, nous avons collecté et modélisé les exigences des décideurs, partant du haut niveau, but, jusqu'à la déduction des mesures, dimensions et attributs que le nouveau DW/DM doit fournir. La phase suivante de notre approche MDA est alors de réaliser le modèle PIM ou modèle conceptuel du DW/DM. En se basant sur le modèle SGAP obtenu et les règles de transformation CIM à PIM automatiques et manuelles, nous obtenons le modèle PIM illustré dans la Figure 3.17.

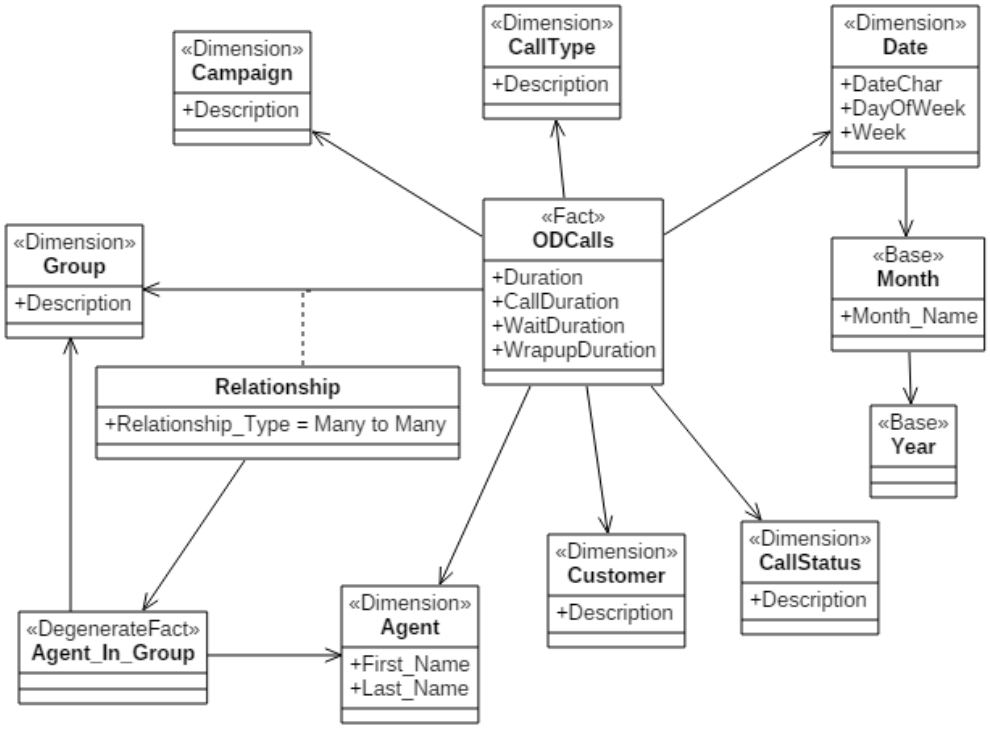


Figure 3.17 Modèle PIM obtenu

Etant donné que toutes les mesures recensées ont la même dimensionnalité, nous obtenons une seule table de fait que nous avons appelé ODCalls, composée des attributs de fait ou mesures recensées dans le modèle SGAP. Le fait obtenu a comme dimension de type un-à-plusieurs: campagne, date, statut (CallStatus), télé-conseiller (Agent est le libellé de télé-conseiller utilisé par l'outil), type (CallType) et client (Customer). Et comme dimension de type plusieurs-à-plusieurs la dimension groupe de télé-conseillers (Group). Chaque dimension est composée des attributs recensés dans le SGAP à l'exception de la dimension Date que nous avons enrichie d'attributs de date utiles pour l'analyse, tels que le jour de la semaine (DayOfWeek), le mois en lettres (MonthName), la semaine (Week) etc.

Une fois le modèle PIM de notre étude de cas est généré, la phase suivante est de générer les modèles PSMs correspondants (OLAP et relationnel). Afin de permettre un passage automatique du PIM vers OLAP, notre approche intègre l'utilisation d'un PDM décrivant la plateforme cible. Etant donné que la plateforme que nous avons utilisé dans cet exemple pour déployer notre cube OLAP est MS Analysis Services, nous avons instancié le métamodèle PDM pour décrire les capacités de cette plateforme comme le montre la Figure 3.18. Ainsi, nous générons le modèle OLAP à partir des deux modèles PDM et PIM résultants, et le modèle relationnel à partir du modèle PIM.

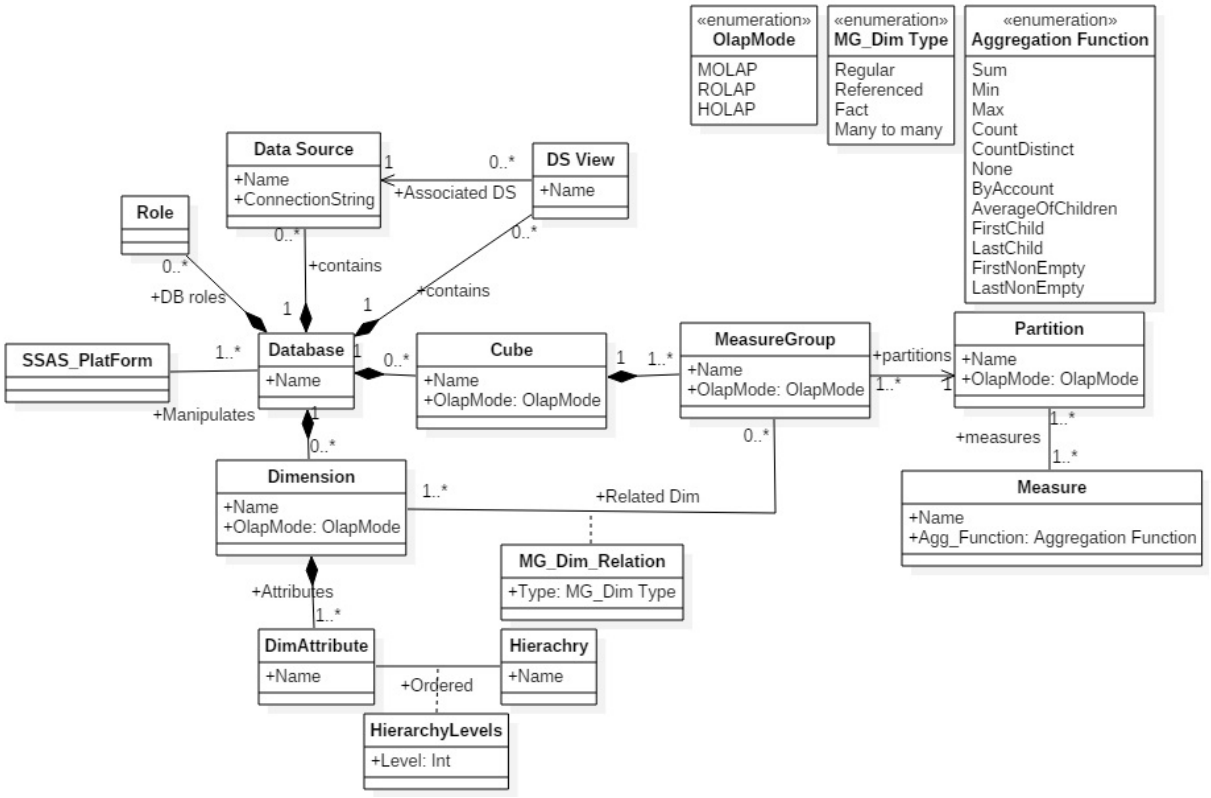


Figure 3.18 Modèle PDM de la plateforme Microsoft SSAS

En arrivant à la dernière étape de notre approche, nous générons le code d'implémentation relatif à la plateforme relationnelle et celle OLAP, via les règles de transformation de type modèle à texte prédéfinies.

Comme pour les transformations de type modèle à modèle, nous utilisons le langage ATL pour implémenter les transformations modèle à texte. En fait le langage ATL, contrairement au langage QVT, ne nécessite pas de packages supplémentaires pour l'implémentation des transformations modèle à texte, il est, par ailleurs, possible de générer du code écrit dans un fichier via la commande ATL query.

Ainsi, nous générons le code SQL de création des tables relationnelles à partir du modèle relationnel, et le code XMLA d'implémentation du cube OLAP à partir des deux modèles, relationnel et OLAP. La sortie de cette phase consiste en deux fichiers « File.sql » et « File.xmla » prêts à être exécutés pour créer les tables et le cube correspondant, respectivement. La Figure 3.19 montre le résultat du cube déployé après l'avoir traité.

Enfin, comme discuté auparavant, notre approche inclut également la collecte des métadonnées afin de les utiliser par la suite pour la documentation du système et le traçage de la lignée des données dans le cadre de la gouvernance des données.

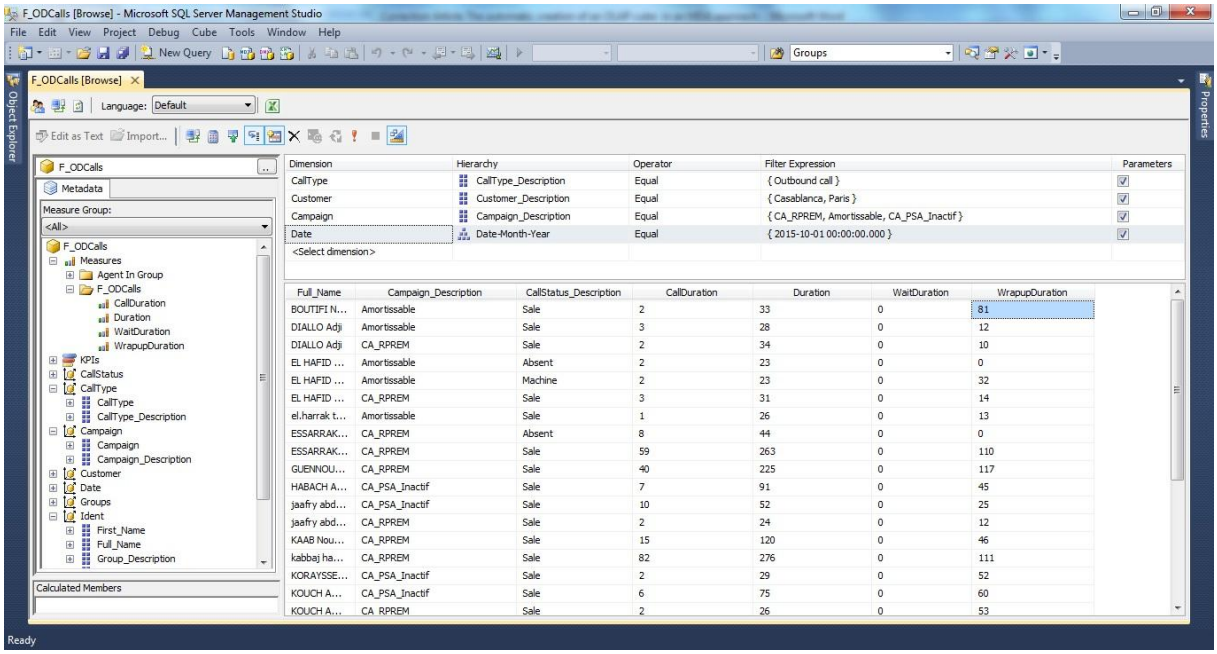


Figure 3.19 Cube résultant de notre étude de cas

Ainsi, à partir du niveau CIM, les métadonnées collectées concernent principalement les descriptions métier des données (mesures) extraites du modèle GoalCases, en plus de la description des sources de données disponible dans le modèle SGAP. La Figure 3.20 affiche une partie des métadonnées collectées à ce niveau.

Par la suite, à partir du modèle PIM, de nouvelles métadonnées techniques décrivant les entités déjà créées (Mesures) sont ajoutées. Enfin dans le niveau PSM, le plus riche en métadonnées techniques, nous extrayons les métadonnées décrivant le DW relationnel, les tables et les colonnes et ce à partir du modèle relationnel. De même, pour le modèle OLAP, les métadonnées décrivant la base OLAP, le cube et dimensions sont répertoriées.

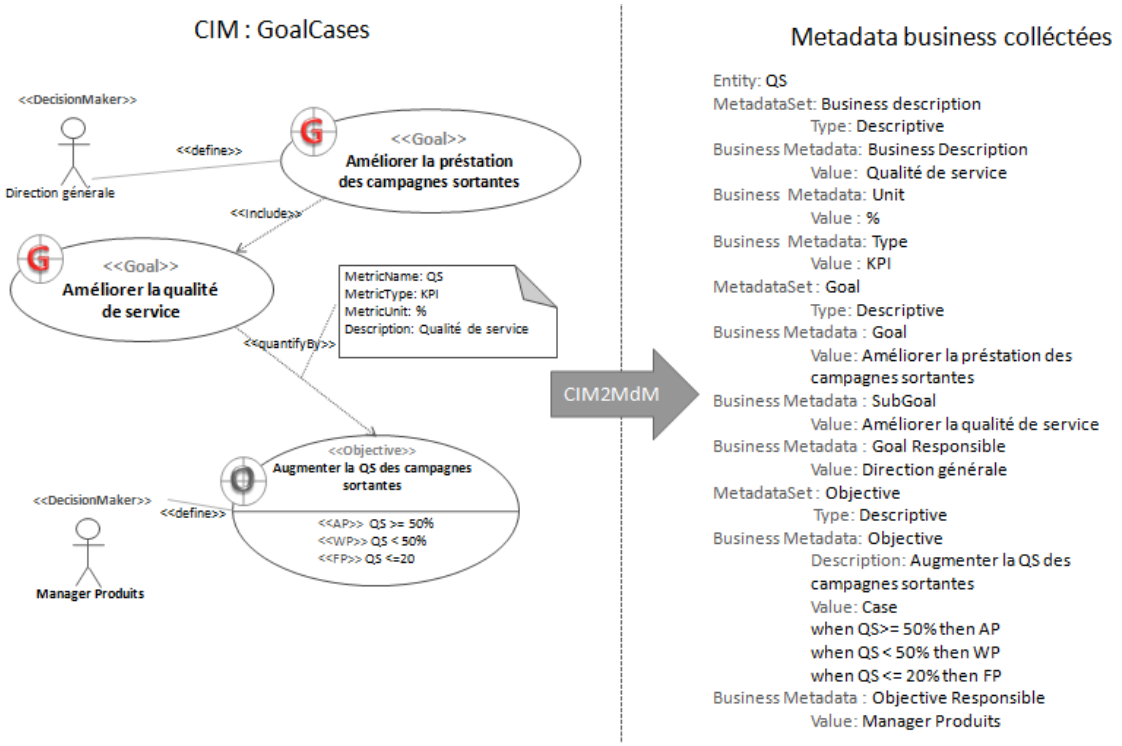


Figure 3.20 Extrait des métadonnées collectées du modèle GoalCases

3.9. Conclusion

L'architecture MDA vise à modéliser et automatiser les applications logicielles. Dans le cas des projets décisionnels, la MDA peut être alors très utile. En effet, le projet décisionnel est un projet itératif dont le développement peut prendre plusieurs années avant d'être achevé. En outre, ses sous projets ou data marts ont toujours la même structure. Ainsi, en utilisant l'approche MDA, nous pouvons réduire le temps et les ressources nécessaires pour le développement et, par conséquent, minimiser le coût du projet. A cet égard, notre approche MDA fournit une démarche, mais aussi un outil pratique pour la mise en place des projets décisionnels qui peut être utilisé par n'importe quel utilisateur même sans connaissance approfondie de la plateforme utilisée.

En fait, contrairement aux nombreux travaux de la littérature qui ont utilisé la MDA dans le développement des projets décisionnels mais sans pour autant la concrétiser en allant jusqu'à l'implémentation des cubes OLAP, notre approche, quant à elle, traite la modélisation des projets décisionnels depuis la définition des besoins utilisateurs jusqu'à la génération du code final. En effet, au niveau de la définition des besoins (CIM), notre approche MDA orientée but, fournit un outil de modélisation simple et adapté aux concepts multidimensionnels, permettant de modéliser progressivement les exigences des décideurs partant des buts stratégiques modélisés avec le GoalCases jusqu'à la déduction des mesures et axes d'analyse via notre modèle SGAP. Ce dernier et grâce à l'utilisation de la notation BPMN, offre une meilleure lisibilité mais aussi une facilité d'extension du modèle CIM. Par la suite, et via un processus de transformation semi-automatique, nous générons le modèle conceptuel (PIM) du nouveau DW/DM conforme à un métamodèle multidimensionnel et qui prend en charge tous les cas de modèles possibles. A partir de ce modèle conceptuel, nous générons deux modèles spécifiques, relationnel et OLAP, les deux sont conformes à leurs métamodèles du standard OIM, un choix que nous avons détaillé dans la section 2.5. Afin d'automatiser ce passage, et vu les particularités de chaque plateforme OLAP, notre approche intègre un modèle PDM décrivant la plateforme OLAP cible. Ce denier modèle précise en particulier si la plateforme cible supporte le type de relation plusieurs-à-plusieurs ainsi que les modes de stockage supportés par cette plateforme. Par la suite, à partir du modèle relationnel, nous générons le code SQL de création des tables du DW ainsi que les différentes contraintes d'intégrité requises. Ensuite, à partir des deux modèles, relationnel et OLAP, nous générons le code d'implémentation du cube OLAP. Pour cela, nous avons traité le cas de la plateforme SQL Server Analysis Services en générant le code XMLA associé. De plus, et afin de profiter de notre architecture MDA et répondre à la problématique courante relative à la documentation et la traçabilité dans les systèmes décisionnels, notre approche inclut la collecte et la sauvegarde automatique des métadonnées disponibles dans chaque niveau de notre architecture.

Enfin, pour valider notre approche, nous avons traité un cas réel relatif à un centre de contacts, pour qui nous avons mis en place un data mart servant à analyser les appels par télé-conseiller, campagne, date et statut d'appel afin de permettre au top management de faire le suivi de leurs buts stratégiques.

Toujours dans le cadre de minimisation du coût global des systèmes décisionnels, nous traiterons dans le chapitre qui suit les techniques d'optimisation de ces derniers visant à améliorer leur

performance et réduire leur coût de maintenance. Nous mettrons l'accent par la suite sur la technique de partitionnement horizontal.

Chapitre 4

Le Partitionnement Horizontal dans les Entrepôts de Données et Cubes OLAP

Résumé : Bien qu'ils soient conçus pour l'analyse de grands volumes de données, grâce à leurs conception et architectures adaptées ainsi que leurs modes de stockage optimisés, les systèmes décisionnels peuvent subir des dégradations importantes des performances, dues à la croissance continue du volume de données. Cette dégradation peut concerner le temps de réponse des requêtes ou le temps de mise à jour du cube, appelé temps de traitement. En outre, les DSS peuvent connaître des problèmes de gestion de l'espace de stockage, en particulier au niveau des cubes OLAP.

Pour pallier à cela, plusieurs techniques existent qu'on peut classer en deux catégories : méthodes redondantes et méthodes non redondantes. Les méthodes redondantes, comme l'indexation des données et les vues matérialisées, permettent d'améliorer considérablement les performances mais elles requièrent, cependant, plus d'espace de stockage. Alors que les méthodes non redondantes comme le partitionnement des données permettent à la fois l'amélioration des performances ainsi que la gestion de stockage. Ces méthodes peuvent être combinées pour profiter des avantages de chacune.

Dans ce chapitre, nous présenterons les méthodes d'optimisation les plus utilisées et nous détaillerons par la suite la technique de partitionnement, objet de notre recherche, ainsi que les approches de l'état de l'art relatives à cette dernière.

4.1. Techniques d'optimisation des DWs et cubes OLAP

En plus du choix d'un mode de stockage approprié (MOLAP pour de meilleures performances et ROLAP pour une optimisation de l'espace de stockage) et d'une architecture adaptée, d'autres techniques existent pour améliorer les performances des DSS. Parmi ces techniques, on trouve ceux qui sont déjà utilisées par les plateformes relationnelles et OLAP, comme les index, que l'administrateur peut toujours reconfigurer, et d'autres, comme le partitionnement, qui doivent être mises en place et définies par l'administrateur.

4.1.1 Les index

Une des méthodes les plus utilisées dans les systèmes de gestion des bases de données pour accélérer l'accès aux données est l'indexation. Un index fournit un moyen rapide pour localiser les données. En effet, sans la création d'index, les requêtes doivent, dans la plupart du temps, parcourir séquentiellement la totalité des données, tandis qu'elles effectuent une seule lecture dans le cas d'index approprié. Toutefois, malgré son avantage relatif à l'accès rapide aux données, l'indexation présente aussi des inconvénients. En effet, presque chaque mise à jour de l'attribut indexé nécessite la mise à jour de l'index, ce qui peut impacter les performances du processus de mise à jour du DW. Un autre problème relatif à l'utilisation des index dans les DWs, est celui de la sélection des index, c.à.d le choix de la meilleure configuration d'index et qui offrira les meilleures performances (Aouiche et al, 2005).

Il existe plusieurs techniques d'indexation dont les plus connues sont :

Index B-arbre (B-tree) : est la technique la plus utilisée dans les bases de données relationnelles. En effet, tous les grands éditeurs de SGBD prennent en charge une des variations de cette technique. Dans cette dernière, un index est une structure à multi-niveaux appelée arbre équilibré, contenant un nœud racine et des pointeurs vers les niveaux bas suivants dans l'arbre (Vaisman, 2014). Le niveau le plus bas de cet arbre contient la valeur de la donnée et des pointeurs vers les lignes correspondantes. Cette technique présente cependant la même complexité de recherche que d'update ce qui la rend adaptée aux SGBD relationnels mais moins pour les bases OLAP, du fait que ces dernières requièrent des opérations de recherche complexes et l'extraction de grands volumes de données. Ceci conduirait à des index avec un grand nombre de niveaux, et qui ne sont donc pas très efficaces.

Index Bitmap : contrairement aux index B-arbre, qui pour chaque attribut indexé stocke dans l'index la même valeur que celle dans la table, l'index bitmap crée pour chaque valeur possible de l'attribut un vecteur de bits de même longueur que la table, prenant la valeur 1 si la ligne associée a la valeur du bitmap (Vaisman, 2014). En considérant par exemple un index bitmap créé sur un attribut A d'une table et $\{v_1, v_2, .., v_n\}$ les valeurs possibles de A, alors un vecteur bitmap sera créé pour chacune des valeurs de A. Ainsi, si le i ème bit de v_k a la valeur 1 alors la i ème ligne de la table a la valeur v_k dans la colonne A.

De ce fait, l'index bitmap consomme beaucoup moins d'espace que celui d'un B-arbre et permet une faible consommation du CPU, en particulier si la cardinalité de l'attribut indexé est faible. Il est donc recommandé dans les entrepôts de données et systèmes OLAP.

Index de jointure : Etant donné que les jointures sont l'un des mécanismes les plus coûteux en termes de performance dans les bases de données, les index de jointure ont pour but d'optimiser ces jointures. Le principe de ce type d'index est de pré-calculer la jointure associée et stocker le résultat dans une table à deux colonnes correspondant chacune à une clé primaire de l'une des tables concernées par la jointure (Vaisman, 2014). Ce type d'index peut être utile dans le cas des DSS du fait du grand nombre de jointures entre table de fait et de dimensions.

4.1.2 Les vues matérialisées

Une vue matérialisée est une vue physiquement créée dans la base de données. Ainsi, contrairement aux vues dont la requête associée doit être ré-exécutée à chaque invocation de la vue, les vues matérialisées permettent d'améliorer les performances des requêtes en jouant le rôle d'un cache accessible directement sans la consultation des relations de base. Elles peuvent donc être utilisées pour stocker le résultat de requêtes coûteuses, comme celles contenant des jointures ou des agrégations, mais également pour répondre à des contraintes de sécurité ou de confidentialité (Bellatreche, 2003).

Cette technique a cependant des inconvénients. En effet, lorsque les relations de base sont mises à jour, les vues matérialisées issues de ces relations doivent également être mises à jour, ce qu'on appelle maintenance des vues matérialisées (Vaisman, 2014). Ceci peut augmenter le coût de maintenance du DW. En outre, la création des vues matérialisées suppose la connaissance préalable des requêtes coûteuses et fréquentes, appelée problème de sélection des vues matérialisées. Pour cette raison, plusieurs algorithmes ont été proposés dans la littérature pour traiter la maintenance et la sélection des vues matérialisées (Mami et al, 2012).

Enfin, comme déjà discuté, les vues matérialisées comme les autres solutions redondantes requièrent un espace disque additionnel.

4.1.3 Le partitionnement

Le partitionnement, appelé aussi fragmentation (Vaisman, 2014), est la segmentation des données en unités physiques distinctes pouvant être gérées indépendamment (Inmon, 2005). Ces unités sont appelées chacune une partition. Le partitionnement peut être appliqué aux tables comme aux index. En outre, le partitionnement des index peut s'appliquer à des tables non partitionnées et vice-versa (Vaisman, 2014).

Il existe deux types de partitionnement, vertical et horizontal :

Partitionnement vertical : il consiste en la fragmentation d'une table en plusieurs, composées chacune d'un sous ensemble des colonnes de la table initiale et pouvant être stockées indépendamment (Figure 4.1). Il est donc utile pour le traitement des requêtes de projection portant sur les colonnes utilisées dans le partitionnement de la table, en limitant ainsi le nombre de colonnes chargées dans la mémoire.

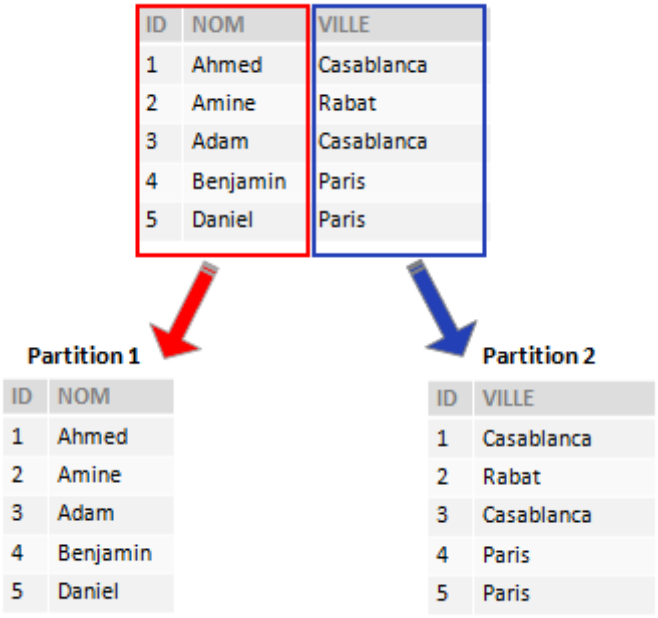


Figure 4.1 Exemple de partitionnement vertical

Partitionnement horizontal : il consiste à diviser une table en des tables plus petites ayant la même structure que la table partitionnée, mais avec moins d'enregistrements (Vaisman, 2014), comme illustré par la Figure 4.2. Ce mode offre, en plus de l'optimisation, plusieurs avantages relatifs au traitement parallèle des requêtes et de mise à jour des cubes OLAP (Kimball, 2013)

ainsi que la gestion de l'espace de stockage et l'archivage automatique des données (Inmon, 2005).

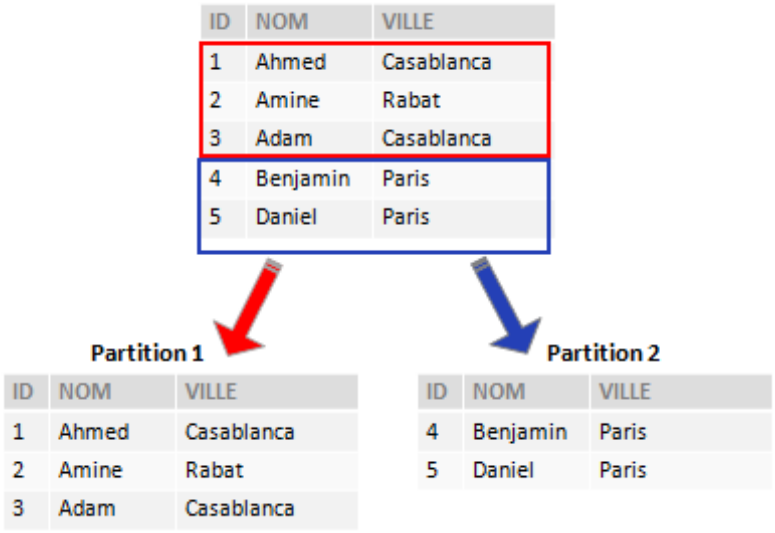


Figure 4.2 Exemple de partitionnement horizontal

Pour toutes ces raisons, il est le mode de partitionnement le plus utilisé et le plus recommandé dans la littérature et par les commerciaux.

Dans ce qui suit, nous mettons l'accent sur cette technique et nous dressons les approches de l'état de l'art relatives aux stratégies de partitionnement.

4.2. Optimisation du DW ou du cube OLAP

La mise en place d'un système décisionnel performant doit tenir compte de plusieurs aspects (Chapitre 1). Il s'agit, en premier lieu, de choisir une architecture adaptée et optimisée. Ensuite, et pour profiter pleinement des avantages des technologies décisionnelles, l'utilisation d'outils OLAP est indispensable, évitant ainsi d'accéder directement au DW via des requêtes SQL peu performantes et peu adaptées à l'analyse multidimensionnelle. Par après, dans la couche OLAP, plusieurs choix s'offrent également. Ainsi, pour obtenir les meilleures performances le choix du mode MOLAP est fortement préconisé, au moment où l'utilisation du mode ROLAP doit être renforcée par l'utilisation des tables d'agrégations permettant de pré-calculer et stocker les agrégations dans la base relationnelle. Enfin, le choix de la plateforme est aussi important étant donné les capacités et limitations de chaque plateforme BI.

Une fois toutes ces décisions post-développement sont prises, des techniques d'optimisation additionnelles peuvent être implémentées comme vu dans la section précédente.

Ainsi, les techniques d'optimisation comme l'indexation, les vues matérialisées ou le partitionnement, appliquées au DW relationnel, peuvent améliorer considérablement le temps de

chargement du cube et de calcul des agrégations. Au moment où, l'utilisation de ces techniques sur le cube OLAP, couche directement interrogée par les utilisateurs, permettra d'améliorer, en plus du processus de chargement, le temps de réponse des requêtes.

4.3. Partitionnement horizontal

4.3.1. Description formelle

Dans le modèle relationnel, considérant une table T caractérisée par un ensemble d'attributs $A = \{a_1, a_2, .., a_k\}$, une partition t de la table T est définie par un ensemble d'attributs. t est notée par $t\langle A_t, y_t\rangle$ où A_t est un sous ensemble de A et y_t est l'ensemble des prédicats permettant de dériver t. En d'autres termes, t est l'ensemble de lignes satisfaisant y_t. y_t est aussi appelé minterm (Ceri et al, 1982).

Dans le cas de partitionnement horizontal, T et t ont la même structure, ce qui signifie que $A_t = A$. Nous pouvons noté alors $t(y_t)$.

On distingue entre partitionnement primaire portant sur un attribut de T et partitionnement dérivé ou référencé portant sur un attribut d'une autre table en relation avec T (Özsu, 2011).

Dans le modèle OLAP, une partition est un sous-ensemble d'un cube créé pour des raisons de performance ou de stockage. Une partition contient toutes les mesures et dimensions utilisées par la partition. Ainsi, une partition horizontale contient toutes les mesures et dimensions de son cube alors qu'une partition verticale contient un sous-ensemble des mesures et dimensions de son cube (MDC-OIM, 1999).

En considérant alors un cube C défini par un ensemble de dimensions D et un ensemble de mesures M, nous pouvons définir une partition P de C par :

P $\langle D_p, M_p, Y_p\rangle$ où D_p est l'ensemble de dimensions de P, M_p est l'ensemble des mesures de P et Y_p l'ensemble des prédicats utilisés pour dériver P.

Dans le cas de partition horizontale, on note P par P(Y_p) vu que $D_p = D$ et $M_p = M$.

4.3.2. Règles de correction du partitionnement

Le partitionnement d'une table ou d'un cube OLAP, doit satisfaire les règles de correction suivantes (Özsu, 2011):

Complétude : Assure que les données d'une relation partitionnée sont mappées vers l'un des fragments sans aucune perte. Autrement dit, en considérant une relation R fragmentée en $F_R = \{R_1, R_2, .., R_n\}$, alors chaque item de R peut être retrouvé dans un ou plusieurs R_i. Dans le cas de partitionnement horizontal, l'item est un n-uplet, alors que dans le cas de partitionnement vertical, l'item est un attribut.

Reconstruction : Assure qu'une relation peut être reconstruite à partir de ses fragments. Ainsi, pour une relation R fragmentée en $F_R = \{R_1, R_2, .., R_n\}$, il existe une opération ∇ telle que :

$R = \nabla R_i , \quad \forall R_i \in F_R$

L'opération ∇ peut être différente selon la forme de fragmentation.

Disjonction : Assure la disjonction des fragments. Ainsi, pour une relation R fragmentée en $F_R = \{R_1, R_2, .., R_n\}$, toute donnée d_i dans R_j ne peut se trouver dans d'autres fragments $R_k (k \neq j)$. Dans le cas de partitionnement vertical, la disjonction concerne tous les attributs à l'exception de la clé primaire qui est répétée dans tous les fragments.

4.3.3. Avantages du partitionnement horizontal

Le partitionnement est l'un des aspects les plus importants dans le développement et la maintenance des systèmes décisionnels. Inmon considère le partitionnement comme la deuxième problématique importante à traiter après celle de la granularité (Inmon, 2005). En fait, le partitionnement offre plusieurs avantages, à savoir :

Maintenance du système : le partitionnement permet de faciliter la maintenance du DW et des cubes OLAP. En effet, au niveau de l'espace de stockage, le partitionnement permet de stocker les partitions sur plusieurs serveurs selon la contrainte de stockage (Vaisman, 2014). En plus, grâce au partitionnement, la stratégie de mise à jour du cube peut être définie selon les besoins des utilisateurs. Par exemple, en créant une partition pour les données du jour, une autre pour les données du mois et une autre pour l'année, la première pourrait être traitée plusieurs fois par jour pour alimenter les rapports journaliers sans affecter les autres partitions, et traiter celle mensuelle chaque soir, et celle annuelle une fois par mois.

Amélioration des performances : le partitionnement permet de réduire le nombre de lignes que le système doit parcourir pour chaque requête utilisateur (Vaisman, 2014). Il permet aussi de mettre en œuvre le parallélisme (Kimball, 2013) pour le traitement des requêtes ainsi que pour la mise à jour du cube. Il est particulièrement utile pour le rafraîchissement des cubes OLAP du fait qu'il permet d'éviter de recalculer à chaque mise à jour toutes les agrégations possibles.

Amélioration de la disponibilité : Un des soucis majeurs dans les systèmes décisionnels est la gestion des problèmes de synchronisation. En effet, la disponibilité du DSS peut être impactée par plusieurs facteurs externes qui peuvent empêcher la mise à jour du système, comme par exemple les coupures d'électricité ou l'indisponibilité des sources de données. Dans ces cas, l'administrateur BI doit être en mesure de récupérer rapidement le cube OLAP pour remettre en ligne les reportings associés, en particulier ceux urgents. Dans le cas de cube non partitionné,

ceci peut prendre plusieurs minutes voir plusieurs heures. Tandis que dans le cas de cube partitionné, l'administrateur BI pourrait commencer par récupérer la partition relative aux données urgentes, ce qui prendra beaucoup moins de temps que de traiter le cube en entier. Ajoutant aussi, la transparence de mise à jour qu'offre le partitionnement grâce à la possibilité de traiter une partition, au moment où les autres restent en ligne, contrairement au cube non partitionné dont le traitement implique la mise hors ligne de toutes les données.

4.3.4. Les stratégies de partitionnement

Bien que le partitionnement soit supporté par la plupart des éditeurs SGBD et OLAP, la définition d'une stratégie de partitionnement reste un vrai défi pour les administrateurs BI. En effet, il n'est pas question de partitionner ou non le DW ou le cube OLAP, mais de comment le faire (Inmon, 2005). Il existe en fait trois stratégies de partitionnement de base :

Partitionnement par rang : il consiste à affecter les enregistrements aux partitions en fonction des plages de valeurs de la clé de partitionnement. Les attributs de la dimension Date sont un candidat naturel pour le partitionnement par rang même si d'autres dimensions peuvent être utilisées (Vaisman, 2014). Par exemple, en utilisant la date comme clé de partitionnement, la partition 012018 contiendra les enregistrements de clé allant du 01/01/2018 au 31/01/2018.

Partitionnement de hachage : se base sur un algorithme de hachage appliqué à la clé de partitionnement. L'algorithme distribue les enregistrements sur les partitions de manière uniforme, résultant idéalement en des partitions de même taille. Ce type de partitionnement est particulièrement utile pour la distribution des partitions entre plusieurs serveurs. L'utilisateur peut alors fournir uniquement la clé de partitionnement et le nombre souhaité de partitions.

Partitionnement par liste : permet de contrôler explicitement la distribution des enregistrements entre les partitions, en spécifiant une liste de valeurs discrètes pour la clé de partitionnement. Par exemple, en utilisant la ville comme clé de partitionnement, nous pouvons créer une partition Ouest définie par la liste des villes {Casablanca, Rabat, Mohammedia}.

Partitionnement composite : est une combinaison des méthodes de partitionnement de base. Ainsi, une table peut être, par exemple, partitionnée initialement par rang, ensuite utiliser le partitionnement de hachage pour diviser d'avantage les partitions résultantes.

4.4. Approches de partitionnement horizontal dans les DWs et cubes OLAP

Etant donné les nombreux avantages d'optimisation et de gestion qu'offre le partitionnement, en particulier dans les entrepôts de données et cubes OLAP, plusieurs travaux dans la littérature ont proposé des approches de partitionnement, parmi lesquelles nous pouvons citer les approches suivantes :

4.4.1. Approche COM-MIN

Elle a été proposée par (Ceri et al, 1982) et se base sur les prédicats des requêtes utilisateurs comme entrée de partitionnement. Son principe est le suivant :

En considérant une relation $R[A_1, A_2, .., A_m]$ et D_i l'ensemble des valeurs de A_i, appelé domaine de A_i, on déduit à partir de la charge de travail, l'ensemble des prédicats simples $Pr=\{p_1, p_2, .., p_n\}$ relatif à R, tel que chaque prédicat peut être noté par : $p_j = A_i \theta\, Valeur$ où $\theta \epsilon \{=, \neq, <, >, \leq, \geq\}$ et $Valeur \in D_i$. Pr doit satisfaire les contraintes suivantes :

- La complétude : signifie que tous les prédicats nécessaires sont pris en compte, pour assurer ainsi que la fragmentation permettra une charge équilibrée entre les fragments.
- La minimalité : signifie que tous les prédicats sont pertinents pour la fragmentation, c'est-à-dire que pour chaque prédicat simple p_k de *Pr*, il existe au moins une application accédant différemment à f_i et f_j (les deux partitions résultantes de la fragmentation utilisant p_k) (Özsu, 2011).

On définit alors l'ensemble des minterms de prédicats M=$\{m_1, m_2, .., m_z\}$ par :

M=$\{m_i | m_i = \bigwedge_{p_k \epsilon Pr} p*_k\},\ 1 \leq k \leq n, 1 \leq i \leq z$

où $p*_k$=p_k ou $p*_k = \overline{p_k}$ (forme négative de p_k). En d'autres termes, M est composé de toutes les conjonctions des prédicats simples de Pr, d'où la cardinalité de M est z=2^n .

Par exemple :

si Pr=$\{p_1, p_2\}$ tel que p_1 : PNom= Développement et p_2 : Budget<3000

Les minterms résultants sont :

m_1 : (PNom= Développement) $\wedge$ (Budget $<$ 3000)

m_2 : Not (PNom= Développement) $\wedge$ (Budget $<$ 3000)

m_3 : (PNom= Développement) $\wedge$ Not (Budget $<$ 3000)

m_4 : Not (PNom= Développement) $\wedge$ Not (Budget $<$ 3000)

Par la suite, l'ensemble M est réduit de façon à éliminer les minterms insignifiants ou contradictoires (Özsu, 2011). Par exemple, en considérant l'ensemble de prédicats $Pr' = \{p_1, p_2\}$

tel que : $p_1: att = val_1$ and $p_2: att = val_2$ et $\{val_1, val_2\}$ sont les valeurs possibles de att, le minterm $m_1: (att = val_1) \wedge (att = val_2)$ est contradictoire car l'attribut att ne peut pas être égal à val_1 et val_2 en même temps. m_1 sera alors ignoré.

Enfin, le schéma de partitionnement retenu correspondra à l'ensemble M réduit.

Cette approche sera la base de plusieurs travaux ultérieurs qui vont tenter de réduire le nombre de minterms de prédicats générés, connu sous le nom de problème de simplification des prédicats (Zhang et al, 1994).

4.4.2. Approche basée sur l'affinité des prédicats

Proposée par (Zhang et al, 1994), elle est inspirée de l'approche de partitionnement vertical proposée par (Navathe et al, 1984), basée sur le calcul de la matrice d'affinité des attributs AAM (Attribut Affinity Matrix).

Son principe est le suivant : après l'énumération des prédicats utilisés dans les transactions ou requêtes importantes, on construit la matrice d'utilisation des prédicats PUM (Predicats Usage Matrix) à l'instar de la matrice d'utilisation des attributs (Navathe et al, 1984). Dans la matrice PUM, chaque ligne représente une requête importante et chaque colonne représente un prédicat simple. La valeur 1 dans une cellule PUM[i,j] indique que la j ème transaction utilise le i ème prédicat (Özsu, 2011), comme illustré par la Figure 4.3.

	P_1	P_2	P_3	P_4
Q_1	1	0	1	1
Q_2	0	1	1	0
Q_3	0	1	1	1

Figure 4.3 Exemple de matrice d'utilisation des prédicats

Par la suite, on construit la matrice d'affinité des prédicats de la taille n x n (où n est le nombre de prédicats) visant à introduire le poids de la fréquence d'accès aux transactions. Pour ce faire, on calcule la somme des nombres d'accès aux transactions accédant simultanément à chaque couple de prédicats.

En utilisant l'exemple de la Figure 4.3, et en supposant que les fréquences d'accès aux requêtes Q_1, Q_2 et Q_3dans une période donnée, sont :

acc(Q_1)=20

acc(Q_2)=10

acc(Q_3)=15

On obtient la matrice d'affinité PAM suivante (Figure 4.4):

	P_1	P_2	P_3	P_4
P_1	-	0	20	20
P_2	0	-	25	15
P_3	20	25	-	35
P_4	20	15	35	-

Figure 4.4 Exemple de matrice d'affinité des prédicats

D'autres valeurs peuvent être utilisées pour indiquer les cas particuliers suivants (Toumi, 2015) :

- * si P_i et P_j se réfèrent au même attribut
- => si P_i implique P_j
- <= P_j implique P_i

On construit, par la suite, les clusters de prédicats selon l'algorithme de partitionnement vertical proposé par (Navathe et al, 1989). Après l'exclusion des zéro, on transforme la matrice d'affinité des prédicats en graphe d'affinité, dont les nœuds sont les prédicats de la matrice d'affinité et les arêtes indiquent l'affinité reliant les deux prédicats en question. L'algorithme, commençant par un nœud aléatoire, tend à chaque itération de construire un cycle de prédicats, représentant une partition candidate, en choisissant le nœud ayant la plus grande affinité. L'algorithme continue ainsi jusqu'à ce qu'il n'y est plus possibilité d'extension du cycle, dans quel cas la partition résultante sera retenue, et ainsi jusqu'à épuisement des prédicats disponibles.

4.4.3. Approche basée sur un modèle de coût

Proposée par (Bellatreche et al, 2000), le principe de cette approche est de sélectionner parmi un ensemble de solutions initiales, le meilleur schéma de partitionnement en se basant sur un modèle de coût. Ce dernier consiste à calculer le nombre d'entrées/sorties (E/S) nécessaires à exécuter les requêtes utilisateurs.

Après l'extraction de la liste des prédicats simples à partir des requêtes utilisateurs, on applique aux domaines de chaque attribut de dimension, l'algorithme COM_MIN (Ceri et al, 1982), afin d'obtenir un ensemble de prédicats minimal et complet. On définit, également, pour chaque

attribut de dimension, le facteur de sélectivité de chacun de ses prédicats c.à.d. le taux qu'il représente par rapport à la table de fait.

Par la suite, en utilisant un algorithme glouton, on sélectionne à chaque itération une dimension pour partitionner la table de fait. On calcule alors le coût du schéma obtenu ainsi que le nombre de partitions résultantes. La dimension est alors retenue pour le partitionnement si le nombre de partitions est inférieur au nombre maximal défini par l'administrateur et que le coût d'E/S est optimal, sinon la dimension est rejetée et l'algorithme passe à une autre dimension.

Pour calculer le coût d'un schéma, le modèle proposé consiste à identifier pour chaque requête Q les fragments requis pour répondre à cette dernière. Ainsi, si on note S=$\{S_1, S_2, .., S_n\}$ l'ensemble des schémas résultant du partitionnement de la table de fait F, on définit la variable suivante :

$$valide(Q, S_j) = \begin{cases} 1 \; si \; le \; sous \; schéma \; S_j \; est \; requis \; pour \; la \; requête \; Q \\ 0 \; sinon \end{cases}$$

Par la suite, on calcule le nombre d'E/S nécessaires pour répondre à la requête Q en utilisant la formule suivante (Bellatreche et al, 2005) :

$$Nombre\_E/S\,(Q) = \sum_{j=1}^{n} valide(Q, S_j) \prod_{i=1}^{M_j} \frac{Sel_F^{p_i} \times |F| \times L}{PS}$$

Où M_j représente le nombre de prédicats de sélection définissant S_j , $Sel_F^{p_i}$ est le facteur de sélectivité du prédicat p_i, |F| est le nombre d'enregistrements de la table de fait, L est la taille d'un enregistrement et PS est la taille d'une page du disque.

Le modèle proposé a été largement utilisé dans la littérature. En effet, les mêmes auteurs (Bellatreche et al, 2005) proposeront plus tard l'utilisation de l'algorithme génétique pour résoudre le problème de partitionnement. La solution proposée consiste à représenter les schémas de partitionnement par des chromosomes auxquels les opérations génétiques (sélection, mutation, croisement) sont appliquées pour générer de nouveaux chromosomes (solutions). Les gènes du chromosome sont les attributs de partitionnement. Par exemple, un gène peut représenter l'attribut catégorie de produit, le mois ou la ville du client, comme illustré par la Figure 4.5. Chaque gène est représenté par un tableau d'entiers de même longueur que le domaine de sélection de l'attribut associé. La solution de partitionnement consiste alors en le produit cartésien des gènes du chromosome. Les allèles d'un gène ayant la même valeur sont regroupés ensemble. La dernière étape consiste alors à définir la fonction de fitness qui va permettre de sélectionner le meilleur chromosome. Pour cela le modèle de coût proposé précédemment est utilisé.

Product category	Books	Sports	Jewelry	Other_Categoies
Gene 1	**1**	**2**	**1**	**3**
Month	January	Other_Months		
Gene 2	**1**	**2**		
Customer state	CO	IL	MN	Other_States
Gene 3	**3**	**2**	**3**	**3**

Figure 4.5 Exemple de chromosome

D'autres algorithmes comme Hill climbing (HC) et Simulated annealing (SA) ont été également utilisés pour le partitionnement, avec le même modèle de coût (Bellatreche et al, 2009).

Plus récemment, (Toumi et al, 2015) ont utilisé le même modèle de coût mais avec cette fois-ci l'algorithme Particle Swarm Optimisation. L'approche proposée se base sur une matrice d'attraction des prédicats, obtenue à partir de la matrice PUM et en utilisant l'indice de Jaccard pour calculer l'attraction entre les prédicats. Ces derniers sont par la suite regroupés en des clusters grâce à l'algorithme Ward. Les partitions sont ensuite définies selon les clusters obtenus et en utilisant l'algorithme Particle Swarm Optimisation pour sélectionner le meilleur schéma de partitionnement.

4.4.4. Approche basée sur le data mining

Dans la littérature, plusieurs algorithmes de data mining ont été proposés pour répondre au problème de partitionnement des entrepôts de données, comme déjà cités. Parmi ces approches, celle proposée par (Mahboubi et al, 2008) appliquant l'algorithme k-means sur la matrice d'utilisation des prédicats pour regrouper les prédicats en des clusters, selon la variable k définie par l'administrateur. Chaque cluster correspond à une partition. Le nombre de partitions créées est alors k+1, en ajoutant la partition contenant les données restantes.

Plus tard, (Hamdi et al, 2015) ont utilisé la même approche en définissant la variable k, cette fois-ci, via un test statistique, visant à vérifier que la distribution actuelle est normale. Ainsi, l'algorithme commence par un k=1, puis augmente sa valeur pas à pas. Dans chaque itération, l'algorithme vérifie si le cluster correspond à une distribution normale, sinon il est partitionné en deux nouveaux clusters. L'approche proposée comprend également un deuxième niveau de partitionnement en temps réel, visant à ajuster les partitions selon les données chargées, en effectuant des opérations de fusion et de partitionnement.

Une autre approche utilisant l'algorithme k-means a été proposée par (Amirat et al, 2014). Elle consiste à diviser les requêtes utilisateurs en deux classes, telles que chacune est affectée à une

méthode d'optimisation, le partitionnement ou l'indexation. Pour classer les requêtes, les auteurs proposent de calculer un poids basé sur la sélectivité des prédicats de la requête. Enfin, pour effectuer le partitionnement, l'algorithme génétique est utilisé. Le tableau 4.1 récapitule les travaux de l'état de l'art présentés :

Tableau 4.1 Récapitulatif des approches de partitionnement de l'état de l'art

Approche	Algorithme de partitionnement	Règle de partitionnement	Domaine	Portée
(Ceri et al, 1982)	Algorithme COM_MIN	Minterm de prédicats	Bases de données relationnelles	Tables
(Zhang et al, 1994)	Algorithme basé sur l'affinité des prédicats	Matrice d'affinité des prédicats	Bases de données relationnelles	Tables
(Bellatreche et al, 2000)	Modèle de coût/Algorithme génétique	Prédicats des requêtes fréquentes	Data warehouse relationnel	Fait et dimension
(Mahboubi et al, 2008)	Algorithme K-means	Matrice PUM	Data warehouse XML	Fait et dimension
(Toumi et al, 2015)	Jaccard index, Algorithme Ward, et particle swarm optimization	Clusters de prédicats basés sur la matrice PUM	Data warehouse relationnel	Fait et dimension
(Hamdi et al, 2015)	Algorithme 2LPA-RTDW : 1-Partitionner les données selon les requêtes utilisateurs 2- Ajuster les partitions selon les nouvelles données chargées	Requêtes utilisateurs (matrice PUM) et taille des données	Data warehouse relationnel temps réel	Fait et dimension
(Amirat et al, 2014)	Classification des requêtes basée sur le k-means pour appliquer une méthode d'optimisation: partitionnement/indexation	Requêtes utilisateurs	Data warehouse relationnel	Fait et dimension

4.5. Conclusion

Le partitionnement est l'une des méthodes d'optimisation des entrepôts de données et cubes OLAP les plus recommandées grâce à ses nombreux avantages. En effet, en plus de l'optimisation du temps de réponse, il permet la définition d'une stratégie de rafraîchissement du DW et cube OLAP et la réduction du délai de ce traitement. Il est alors supporté par la majorité des SGBD et plateformes OLAP. Le défi est, toutefois, de définir la meilleure stratégie de partitionnement.

Plusieurs approches de partitionnement basées sur les requêtes utilisateurs ont été proposées dans la littérature, en particulier dans le cas des bases de données relationnelles. Cependant, toutes ces approches prennent en considération uniquement les requêtes des utilisateurs sans considérer le volume des données et la taille des partitions. Ceci peut engendrer des partitions de tailles trop irrégulières, empêchant ainsi, la gestion de l'espace de stockage en plus de la difficulté de reprise progressive en cas de panne. De plus, dans l'approche proposée par (Ceri et al, 1982), la complexité du problème est de l'ordre de 2^n, chose que les approches ultérieures ont tenté de réduire, mais le nombre de partitions obtenues reste toujours élevé. D'autres approches, comme celle génétique de (Bellatreche et al, 2009), nécessitent la définition de paramètres compliqués, ou l'utilisation de plusieurs techniques complexes comme dans le cas de l'approche proposée par (Toumi et al, 2015).

Dans ce qui suit, nous présentons une nouvelle approche de partitionnement qui prend en considération à la fois le besoin des utilisateurs à travers la fréquence des requêtes utilisateurs et leur criticité, ainsi que le volume des données et la taille des partitions résultantes. La méthode proposée est basée sur l'algorithme des règles d'association.

Chapitre 5

Notre approche de partitionnement OLAP

Résumé : Le partitionnement est une méthode d'optimisation utilisée dans les projets décisionnels pour améliorer les performances des requêtes et de rafraîchissement. C'est pourquoi la plupart des éditeurs BI intègrent cette fonctionnalité. Cependant, ces derniers ne fournissent pas de stratégies de partitionnement, ce qui demeure un sérieux défi pour les administrateurs BI. Comme vu dans le chapitre précédent, plusieurs travaux dans la littérature ont proposé des algorithmes et des stratégies de partitionnement. Néanmoins, la plupart d'entre eux se sont focalisés sur les entrepôts de données relationnels et ont ignoré les cubes OLAP, bien que ces derniers constituent la première couche sollicitée par les requêtes multidimensionnelles des utilisateurs finaux. Pour cela, nous proposons une nouvelle approche de partitionnement horizontal, appelée OPAR (OLAP Partitioning based on Association Rules), basée sur les concepts des règles d'association et qui tient compte à la fois de la fréquence et la criticité des requêtes utilisateurs ainsi que du volume de données et de la taille des partitions résultantes. L'approche proposée a pour objectif d'optimiser le temps de réponse des requêtes et de rafraîchissement des cubes, mais aussi de faciliter la gestion de l'espace de stockage.
Pour évaluer et valider notre approche, nous avons mené un ensemble d'expérimentations dont les résultats sont favorables comparés aux approches existantes.

5.1. Introduction

La majorité des approches de partitionnement précédentes concernent les entrepôts de données relationnels. Cependant, lorsque le système décisionnel intègre une couche OLAP pour interroger le DW afin de répondre aux requêtes utilisateurs, le partitionnement relationnel seul n'est pas la solution. En fait, comme vu dans le chapitre 1, dans une telle architecture, les données sont extraites du DW et sont calculées et agrégées par le serveur OLAP (Ponniah, 2004). Elles sont stockées dans des tableaux dans le cas du mode MOLAP et dans des tables d'agrégations relationnelles dans le cas du mode ROLAP. La stratégie de partitionnement doit donc concerner la couche OLAP. A cet égard, notre approche s'intéresse au partitionnement des cubes OLAP pour améliorer le temps de réponse des requêtes utilisateurs ainsi que le temps de rafraîchissement des cubes (AlHammad, 2016). D'autre part, même si le partitionnement est pris en charge par la plupart des éditeurs OLAP, la définition d'une stratégie de partitionnement appropriée ne peut pas se faire par les outils. En fait, une telle stratégie doit prendre en compte plusieurs paramètres dont le volume des données et le besoin des utilisateurs, notamment la fréquence et la priorité de leurs requêtes.

Pour cela, notre approche de partitionnement tient compte de toutes ces exigences (Letrache et al, 2018). Ainsi, lors de la première phase de notre approche, les requêtes des utilisateurs sont collectées et analysées, pour déduire par la suite les itemsets fréquents et ordonnés de prédicats. Pour ce faire, nous utilisons l'algorithme des règles d'association, apriori. Toutefois, avant d'exécuter ce dernier, une étape de préparation des requêtes est requise, concernant en particulier le traitement des prédicats de type date et l'identification des requêtes critiques. Par la suite, l'algorithme de partitionnement, prenant en entrée les itemsets résultants de la phase précédente, définit progressivement le meilleur schéma de partitionnement. Cet algorithme, basé sur les concepts des règles d'association, fouille dans les données multidimensionnelles, vérifiant la cooccurrence des prédicats des itemsets déduits de la phase d'analyse, en se basant sur la contrainte de support (fréquence). Ainsi, les itemsets satisfaisant cette contrainte participeront au partitionnement du cube, résultant en des fragments multidimensionnels appelés partitions. Pour définir ce seuil de support (min_sup), on considère la contrainte de taille maximale des partitions définie par l'administrateur. Ces deux contraintes, support et taille, permettent d'obtenir un nombre de partitions gérable et dont les tailles sont régulières. Le résultat de ce processus de partitionnement se traduit par l'amélioration des performances du cube et la facilité de gestion de l'espace de stockage et de rafraîchissement ainsi que de reprise en cas de pannes.

Enfin, l'algorithme proposé peut être exécuté périodiquement suite à la constatation d'une dégradation des performances comme illustré par la Figure 5.1.

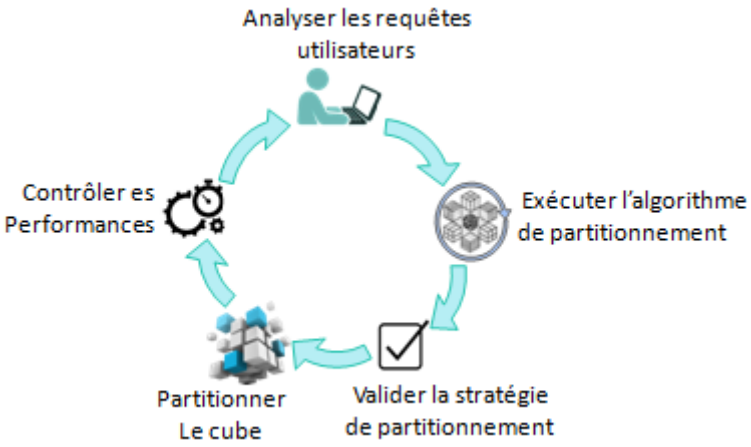


Figure 5.1 Notre approche de partitionnement OLAP

5.2. Définitions

5.2.1. Partition OLAP

Dans le modèle OLAP, une partition est un sous-ensemble d'un cube créé pour des raisons de performance ou de stockage. Une partition contient toutes les mesures et dimensions utilisées par la partition. Ainsi, une partition horizontale contient toutes les mesures et dimensions de son cube, alors qu'une partition verticale contient un sous-ensemble des mesures et dimensions de son cube (MDC-OIM, 1999).

En considérant alors un cube C défini par un ensemble de dimensions D et un ensemble de mesures M, nous pouvons noter une partition P de C par :

P $\langle D_p, M_p, Y_p \rangle$ où D_p est l'ensemble de dimensions de P tel que $D_p \subseteq D$, M_p est l'ensemble de mesures de P tel que $M_p \subseteq M$ et Y_p l'ensemble de prédicats utilisés pour dériver P.

Dans le cas de partition horizontale, on note P par P(Y_p) vu que $D_p = D$ et $M_p = M$.

5.2.2. MultiDimensional eXpressions (MDX)

Le MDX est un langage puissant, qui permet de créer et d'interroger des objets multidimensionnels. Il fournit des commandes pour récupérer et manipuler des données multidimensionnelles à partir de ces objets. La syntaxe MDX est similaire au langage SQL (Structured Query Language), mais ses fonctionnalités peuvent être plus complexes et plus robustes que celles du SQL.

Parmi les objets multidimensionnels manipulés par le MDX, on retrouve les Tuples. Un tuple est une tranche d'un cube (slice). Il est formé d'une combinaison de membres de dimension, de telle façon qu'il n'y est pas deux membres ou plus appartenant à la même hiérarchie (Spofford et al, 2006). Un tuple peut être vu aussi comme un croisement ou un vecteur de membres de données d'un cube. Ainsi, considérant un cube C et ses dimensions D=$\{D_1, D_2.., D_d\}$ et MB_i l'ensemble des membres des niveaux de D_i, on définit un tuple par :

$T=mb_1 \otimes mb_2 .. \otimes mb_n$ où $mb_i \epsilon\ MB_i$

Par exemple :

T=([Date].[Year].[2015],[Customer].[Nation].[France]) est un tuple composé des deux membres *[Date].[Year].[2015]* et *[Customer].[Nation].[France]*.

Un tuple peut contenir un seul membre, dans ce cas il est appelé tuple singulier.

On appelle aussi un Set une collection de tuples (Spofford et al, 2006), tel que $S=\{T_1, T_2.., T_k\}$ où T_i est un tuple.

Par exemple :

S={([Date].[Year].[2015],[Customer].[Nation].[France]),([Date].[Year].[2016],[Customer].[Nation].[Morocco]) }.

5.2.3. Algorithme des règles d'association

Une règle d'association est une implication de la forme X→Y, qui signifie que toutes les transactions dans la base de données qui contiennent X tendent à contenir Y. la déclaration formelle d'une règle d'association est la suivante :

Soit I={$I_1, I_2, .., I_n$} un ensemble d'items. Un ensemble d'items X ⊂ I est appelé itemset. Soit T un ensemble de transactions où chaque transaction t dans T est un itemset tel que t ⊆ I. on appelle règle d'association X→ I_j où X est un ensemble d'items de I, tel que X ⊂ *I* et I_j est un item singulier de I qui n'est pas présent dans X, tel que X ∩ I_j=∅ . X→ I_j est satisfaite avec un facteur de confiance (min_conf) 0≤ c ≤ 1, si au moins c% des transactions dans T satisfaisant X, satisfont aussi I_j (Agrawal et al, 1993).

Une telle règle devrait aussi vérifier les deux contraintes suivantes (Agrawal et al, 1993):

Contrainte syntactique : c'est une restriction sur les items pouvant apparaître dans la règle.

Contrainte de support : elle concerne le nombre de transactions dans T qui supporte une règle. Le support d'une règle peut être défini comme étant la proportion des transactions dans T satisfaisant l'union des items de X et I_j. Il correspond à une signification statistique permettant

de considérer uniquement les règles dont le support est supérieur à un certain seuil minimum (min_sup), sinon elles ne mériteront pas d'être considérées ou sont, tout simplement, moins préférées.

Pour calculer le support d'une règle X→Y dans une base de transactions T, la formule est :

Support(X→Y)= $\frac{Count\ (X \cup Y)}{|T|}$ où |T| est la cardinalité de T.

Et celle de la confiance est :

Confiance(X→Y)=$\frac{Count\ (X \cup Y)}{Count\ (X)}$

Dans ce qui suit, on notera simplement Sup(X,Y) et Conf(X,Y).

On appelle itemset fréquent, tout itemset dont le support est supérieur au min_sup et on note k-itemset tout itemset composé de k items.

5.3. Phase 1 : Analyse des requêtes utilisateurs

Les prédicats des requêtes utilisateurs peuvent être semblables à ceux d'un caddie. Plusieurs corrélations peuvent exister entre les prédicats, et qui ne peuvent pas être déduites des données mais des requêtes eux-mêmes. En les analysant via un algorithme de data mining, nous pouvons déduire des résultats intéressants.

Ainsi, la première étape de notre proposition est de collecter les requêtes utilisateurs à partir des logs du système OLAP. Ces derniers peuvent avoir différents formats (text, csv, table relationnel etc.) selon la plateforme utilisée, mais qui nécessitent généralement un traitement texte. Il existe aussi des outils, comme SQL Profiler (MS-SSP, 2016) et Jmeter (Apache-Jmeter, 2004), qui permettent de tracer les requêtes utilisateurs dans un format simple, mais aussi de fournir plusieurs informations utiles comme l'heure d'exécution de la requête, sa durée, l'utilisateur et le cube associé.

Une fois les requêtes utilisateurs collectées, nous utilisons l'algorithme apriori pour déduire les itemsets fréquents de prédicats. Les itemsets résultants seront alors utilisés comme entrée de l'algorithme de partitionnement lors de l'étape suivante.

Avant d'appliquer l'algorithme apriori, nous devons d'abord prétraiter les requêtes utilisateurs en remplaçant, en premier lieu, les prédicats de type date par leurs formules dynamiques. En effet, lorsqu'un itemset contient un prédicat de type date, alors il ne pourra jamais être considéré comme itemset fréquent car l'item de type date n'a pas une valeur fixe. Toutefois, en comparant

le prédicat date à la date d'exécution de la requête, nous pouvons déduire la corrélation entre les requêtes.

Dans le Tableau 5.1, le prédicat [Date].[Year-Month-Date].[Year].[2016] comparé à la date d'exécution de la requête correspondante, se réfère à l'année précédente. Il sera alors remplacé par la formule MDX dynamique ParallelPeriod([Date].[Year-Month-Date].[Year],1, StrToMember ('[Date].[Year-Month-Date].[Year].['+Year(Now())+']')) et qui retourne l'année précédente à la date en cours. Ainsi, le prédicat année-1 sera commun à tous les itemsets contenant ce dernier. Notant que la fonction StrToMember sert à convertir une chaîne de caractères en un membre MDX et la fonction ParallelPeriod retourne la période parallèle au membre courant selon le paramètre relatif au niveau hiérarchique (année, mois, jour etc.).

Tableau 5.1 Exemples de traitement des prédicats de type date

Prédicat	**Date d'exécution de la requête**	**Prédicat utilisé**
[TIME].[DATE-MONTH-YEAR].[YEAR].&[2016]	21/03/2017	ParallelPeriod([TIME].[DATE-MONTH-YEAR].[YEAR],1,STRTOMEMBER('[TIME].[DATE-MONTH-YEAR].[YEAR].&['+cstr(Year(Now()))+']'))
[TIME].[DATE-MONTH-YEAR].[MONTH].&[20171]	13/01/2017	STRTOMEMBER('[TIME].[DATE-MONTH-YEAR].[MONTH].&['+cstr(Year(Now()))+ cstr(Month(Now()))+']')
[TIME].[DATE-MONTH-YEAR].[DATE].&[20/01/2017]	20/01/2017	STRTOMEMBER('[TIME].[DATE-MONTH-YEAR].[DATE].&['+cstr(Now()))+']'))

L'étape suivante concerne le traitement des opérateurs logiques. En fait, dans le modèle multidimensionnel, la syntaxe générale d'une requête Q_i dans un cube C est :

$\langle$ Select S_i from C where Z_i $\rangle$ tel que S_i est la liste des niveaux de dimensions séléctionnés (projection), C est le cube source et Z_i est l'ensemble des prédicats (selection).

On note alors Q_i par $\langle S_i|C|Z_i \rangle$, Z_i peut être un tuple ou un set de tuples, ce qui insinue l'opérateur logique « OR ». On note alors Z_i par :

$Z_i = \bigcup_1^n T_j$ où T_j est un tuple de Z_i

Chaque tuple T_j dans Z_i représente un itemset de prédicats. Avant de traiter alors la requête Q_i, nous la séparons en plusieurs requêtes Q_j correspondant chacune à un tuple T_j, telle que Q_j est définie par $\langle S_i|C|T_j \rangle$.

Par exemple, si dans une requête Q, nous avons la condition suivante :

Z={([Date].[Year].[2015],[Product].[Name].[Coat]),([Date].[Year].[2016],[Product].[Name].[Jeans])}
alors elle sera divisée en deux requêtes, correspondant respectivement aux tuples :
Z1=([Date].[Year].[2015],[Product].[Name].[Coat])
et Z2=([Date].[Year].[2016],[Product].[Name].[Jeans])

La dernière tâche de préparation concerne la définition de criticité des requêtes. En fait, se baser uniquement sur la fréquence des requêtes, ne reflète pas toujours le besoin des utilisateurs, car il pourrait y avoir des requêtes qui ne sont pas fréquentes (par exemple, celles utilisées par peu d'utilisateurs comme le top management) mais qui sont jugées critiques et requièrent alors de bonnes performances. Pour les traiter, nous intégrons un nouveau paramètre, que nous appelons facteur de criticité permettant d'augmenter la fréquence des requêtes critiques. Le nombre des requêtes utilisateurs T deviendra alors :

Count(T)=$\sum_{i=1}^{n} \alpha_i \, count(Q_i)$ où n est le nombre des requêtes utilisateurs et α_i est le facteur de criticité de Q_i. Notant que α_i a la valeur 1 par défaut.

Une fois que les requêtes utilisateurs et prédicats sont traités, nous exécutons l'algorithme apriori (Han, 2011) afin de calculer la fréquence de chaque itemset de prédicats et identifier les itemsets fréquents (Algorithme 1). Pour ce faire, nous choisissons une valeur très basse pour le min_sup pour laisser l'algorithme de partitionnement, plus tard, contrôler la condition d'arrêt. L'algorithme trie enfin les prédicats de chaque itemset par fréquence ainsi que l'ensemble des itemsets résultants.

Algorithme 1 : Algorithme Apriori pour la génération des itemsets fréquents de prédicats

Input : transactions : *Liste des transactions extraites des requêtes utilisateurs*
items : *Liste des prédicats simples extraite des transactions*
min_sup : *seuil support minimum*

Output allFrequentItemsets: *liste des itemsets fréquents de prédicats*

Procedure AprioriResult(min_sup, items, transactions)

```
frequentItems = GetFrequentItemsNiv1(minsup, items, transactions);// générer les 1-itemsets fréquents
allFrequentItemsets.AddItems(frequentItems); //insérer les 1-itemsets fréquents
transactionsCount = transactions.Count();
do
{
    candidates = GenerateCandidates(frequentItems); // générer tous les candidats possibles à partir de la liste des itemsets fréquents
    frequentItems = GetFrequentItems(candidates, minsup, transactions);// sélectionner les
```

```
itemsets fréquents selon le seuil min_sup
            allFrequentItems.AddItems(frequentItems);
        }
        while (candidates.Count != 0); // tant qu'il existe des itemsets candidats
        allFrequentItems.Sort(); // trier la liste finale d'itemsets par support
return allFrequentItems;
//générer les itemsets de niveau 1 c.à.d les 1-itemsets
Procedure  GetFrequentItemsNiv1(minsup, items, transactions)
          transactionsCount = transactions.Count();
                Foreach (item in items)
                {
                   support = GetSup(item, transactions); //calculer le support de l'item
                   IF (support / transactionsCount >= minSupport)
                         frequentItemsL1.AddItem(item); //ajouter l'item à la liste des itemsets fréquents
                  }
                frequentItemsL1.Sort(); // Trier les itemsets par support
return frequentItemsL1;
Procedure GenerateCandidates (frequentItems)
   For (i = 0; i < frequentItems.Count - 1; i++){
        firstItemSet = frequentItems[i];
        For (j = i + 1; j < frequentItems.Count; j++){
            secondItemSet = frequentItems[j];
               //Insérer l'itemset composé de l'union distincte des items de firstItemSet et secondItemSet
               generatedCandidate.AddDistinct(firstItemSet,secondItemSet);
           IF (generatedCandidate.Count != 0)
              candidates.AddItem(generatedCandidate);
        }
    }
return candidates;
```

5.4. Phase 2 : Algorithme de partitionnement

Notre algorithme de partitionnement (Algorithme 2) vise à fragmenter le cube OLAP en des partitions plus petites, afin d'améliorer le temps de réponse des requêtes et de rafraîchissement du cube. Pour ce faire, notre algorithme, basé sur les concepts des règles d'association, parcourt

d'abord les itemsets fréquents et ordonnés I={$I_1, I_2, .., I_n$} triés par fréquence et résultant de la phase d'analyse des requêtes utilisateurs. Par la suite, il parcourt les partitions P={$P_1, P_2, .., P_m$}, ou le cube dans la première itération, et calcule le support des nouvelles partitions (voir Figure 5.2).

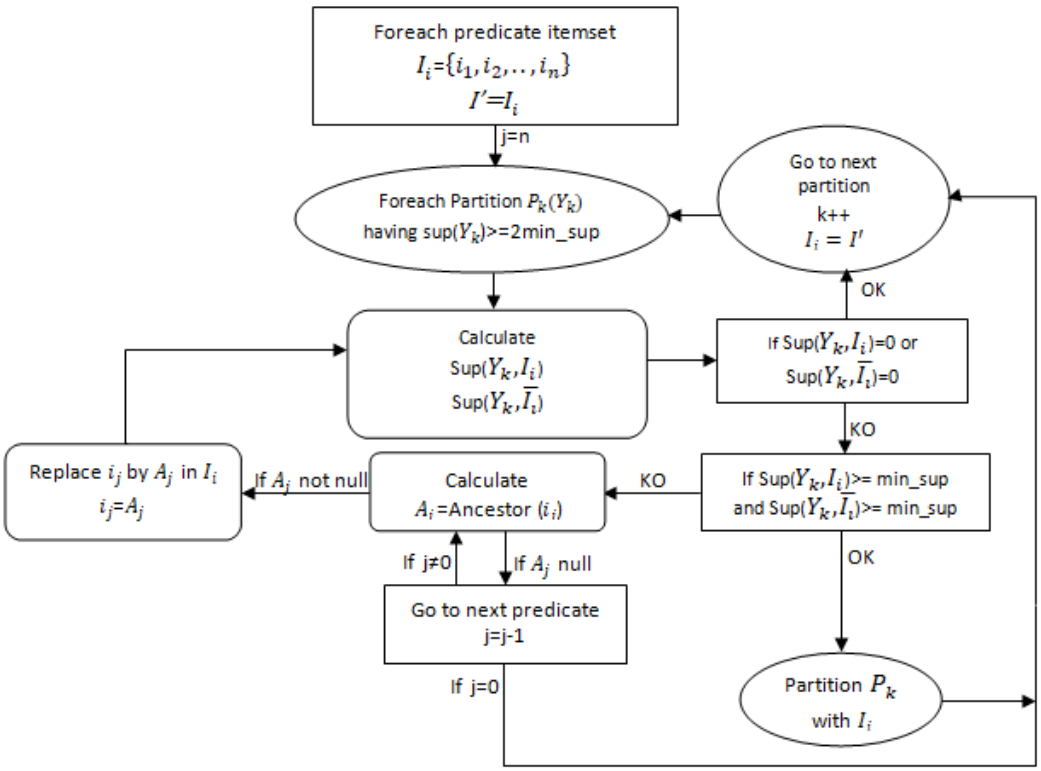


Figure 5.2 Notre algorithme de partitionnement

En fait, en utilisant un itemset $\boldsymbol{I_j}$ pour fragmenter une partition $\boldsymbol{P_i(y_{p_i})}$, nous obtenons deux nouvelles partitions $\boldsymbol{P_{i_1}}$ et $\boldsymbol{\overline{P_{i_1}}}$ qui remplaceront $\boldsymbol{P_i}$, telle que $\boldsymbol{P_{i_1}}$ vérifie $\boldsymbol{I_j \cup y_{p_i}}$ et $\boldsymbol{\overline{P}_{i_1}}$ vérifie $\boldsymbol{\overline{I_j} \cup y_{p_i}}$, comme illustré par la Figure 5.3. Nous avons alors $\boldsymbol{P_{i_1} \cup \overline{P_{i_1}} = P_i}$ et $\boldsymbol{P_{i_1} \cap \overline{P_{i_1}} = \emptyset}$ (règles de correction).

Pour éviter alors de créer un grand nombre de partitions, ou des partitions de tailles irrégulières, et ainsi augmenter le coût de maintenance, l'algorithme calcule les supports de P_{i_1} et $\overline{P_{i_1}}$. Si les valeurs obtenues sont supérieures au seuil min_sup alors l'itemset I_j sera utilisé pour fragmenter la partition P_i.

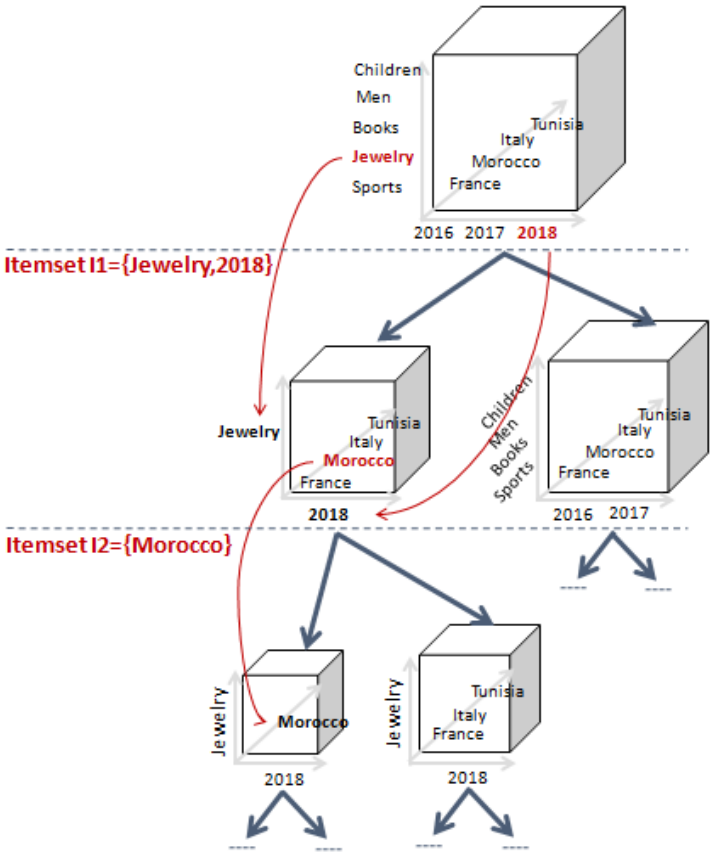


Figure 5.3 Illustration d'un exemple de partitionnement

En fait, le support d'un itemset I dans une partition $P(Y_p)$ est l'occurrence de $Y_p \cup I$ dans le cube C. Pour le calculer, nous utilisons les agrégations pré-calculées de la mesure Count (Messaoud et al, 2006), tel que :

$Sup(Y_p, I) = \frac{Count(Y_p \cup I)}{|C|}$ où |C| est la cardinalité de C telle que |C|=count(All).

Considérant par exemple un itemset I_j={CurrentMonth,Consignment,CustomerCatA} à utiliser pour fragmenter une partition contenant les données du Maroc notée par P_i({Morocco}). Les deux partitions résultantes sont P_{i_1}({CurrentMonth,Consignment,CustomerCatA,Morocco}) et $\overline{P_{i_1}}$({$\overline{\text{CurrentMonth, Consignment, CustomerCatA}}$,Morocco}). Les supports de ces deux partitions et dont les formules sont respectivement :

$$\frac{Count(\{\text{CurrentMonth ,Consignment ,CustomerCatA ,Morocco }\})}{Count(All)}$$

et $$\frac{Count(\{\overline{\text{CurrentMonth ,Consignment ,CustomerCatA}} \text{ ,Morocco }\})}{Count(All)}$$

doivent vérifier le seuil min_sup.

Autrement, l'algorithme tente d'élargir la portée de l'itemset en remplaçant les prédicats par leurs ancêtres dans la hiérarchie associée. L'algorithme commence alors par le dernier prédicat,

représentant en fait celui ayant le plus petit support. Il le remplace par son premier ancêtre et ainsi de suite, jusqu'à satisfaire la contrainte min_sup ou jusqu'à atteindre la racine de la hiérarchie du prédicat, dans quel cas le prédicat sera ignoré. L'algorithme passe alors au prédicat suivant.

Dans le même exemple cité auparavant, en supposant que les supports des deux partitions P_{i_1} et $\overline{P_{i_1}}$ ne vérifient pas la contrainte min_sup, l'algorithme remplace le prédicat "Customer Cat A" par son premier ancêtre et qui correspond à la racine de la hiérarchie notée par "All". Ce qui signifie que le prédicat sera ignoré, et l'itemset I_j deviendra :

I_j=({CurrentMonth,Consignment,All})=({CurrentMonth,Consignment}).

Si les supports ne satisfont toujours pas le min_sup, alors I_j deviendra :

I_j =({CurrentMonth, Jewelry}) comme illustré par la Figure 5.4:

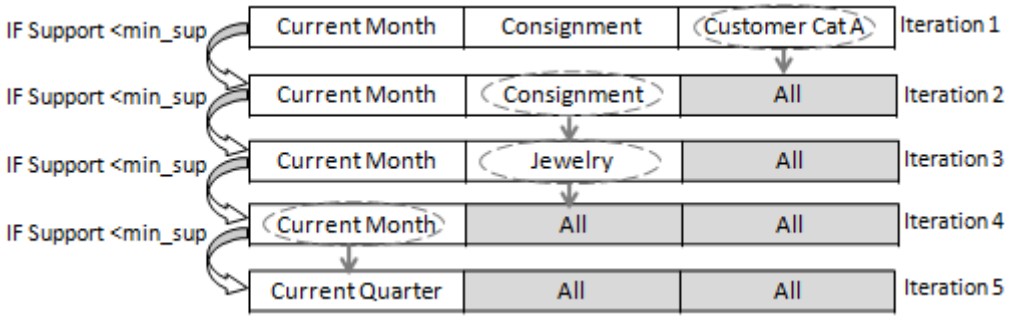


Figure 5.4 Illustration du mécanisme d'élargissement de la portée des itemsets de prédicats

Il faut noter que l'algorithme de partitionnement n'utilise pas de seuil min_conf car le but est d'identifier les itemsets de prédicats qui représentent le meilleur critère de partitionnement et qui constituent une population considérable, plutôt que d'étudier la force de la relation entre les itemsets.

Le nombre maximal de partitions créées est N=2^n où n est le nombre d'itemsets fréquents de prédicats.

D'une autre part, si les supports des nouvelles partitions sont tels que $Sup(Y_{p_i}, I_j)$=0 et

$Sup(Y_{p_i}, \overline{I_j})$=0 alors I_j n'est pas intéressant dans $P_i(Y_{p_i})$. L'algorithme passe alors à l'autre partition.

En outre, si le support de P_i est tel que :

$$Sup(P_i) = Sup(Y_{p_i}) = \frac{Count\ (Y_{p_i})}{|C|} < 2\text{min\_sup}$$

Alors P_i ne pourra pas être divisée en deux nouvelles partitions qui satisfont toutes les deux la contrainte min_sup, car cela impliquera que :

$Sup(Y_{p_i}, I_j)$>=min_sup et $Sup(Y_{p_i}, \overline{I_j})$>=min_sup et ainsi $Sup(Y_{p_i})$>=2min_sup.

L'algorithme passe dans ce cas là aussi à une autre partition.

Enfin, et comme discuté auparavant, le partitionnement sert aussi à gérer l'espace de stockage. Pour cette raison, notre approche permet à l'administrateur du système de fixer une taille maximale (max_size) pour les partitions à créer. Cette taille maximale servira à estimer la valeur appropriée du seuil min_sup.

En fait, la taille d'une partition P_i d'un cube C peut être définie par (Shukla et al, 1996) :

$$Size(P_i) = \frac{|P_i|}{|C|} Size(C) \qquad (1)$$

Et $Size(C) = \sum_1^n Size(P_i)$ où n est le nombre de partitions du cube C.

Si la taille de la partition ne doit pas dépasser un seuil max_size, alors, il faut que :

$$Size(P_i) \leq max\_size \qquad (2)$$

(1) et (2) implique que la condition suivante soit vérifiée :

$$\frac{|P_i|}{|C|} \leq \frac{\text{max\_size}}{Size\,(C)}$$

Il faut alors que :

$$Sup(P_i) \leq \frac{\text{max\_size}}{Size\,(C)} \qquad (3)$$

D'autre part, pour arrêter la fragmentation d'une partition $P_i(Y_{p_i})$, deux cas existent. Le premier, discuté avant, et qui représente le cas idéal, correspond à :

$$Sup(P_i) < 2\text{min\_}sup \qquad (4)$$

Ainsi, en choisissant un min_sup tel que :

$$min\_sup \leq \frac{\text{max\_size}}{2Size\,(C)} \qquad (5)$$

On obtient :

$$Sup(P_i) < \frac{\text{max\_size}}{Size\,(C)}$$

Ainsi, on garantit que (3) et donc (2) seront vérifiées.

Le second cas correspond à l'épuisement de tous les itemsets. Dans ce cas, et même si la condition (5) est vérifiée, la condition (2) ne l'est pas forcément. Cela signifie en fait, que le besoin des utilisateurs est pris en compte mais pas forcément la contrainte de taille, si elle existe. Dans ce cas, un partitionnement par rang selon la dimension date peut être préconisé comme décrit dans la section 5.5.

Algorithme 2 : Notre Algorithme de partitionnement OPAR

Input : C : *cube à partitionner*

min_sup : *le seuil minimum support*

Predicates_Itemsets : *la liste des itemsets fréquents de prédicats*

Output : P *liste des partitions résultantes*

P={C} ; //*P contient initialement le cube*

$\acute{P} = P$; //*sauvegarder l'état de P*

For all $I_i=\{i_1, i_2, .., i_n\} \in$ Predicates_Itemsets **do** //*Parcourir les itemsets fréquents de prédicats*

$I' = I_i$; // *Sauvegarder la valeur initiale de* I_i

$j = n$; //*n est le nombre d'items dans* I_i

$P = \acute{P}$;

For all $p_k(Y_k) \in P=\{p_1, p_2, .., p_m\}$ **do** // *Parcourir les partitions*

IF $\frac{\text{Count}\ (Y_k \cup I_i)}{|C|} \geq 2\text{min\_sup}$ and I_i.descendants.intersect(Y_k) =False **then**

While $j \geq 1$ **do**

$\sup(Y_k, I_i)=\frac{\text{Count}\ (Y_k \cup I_i)}{|C|}$;// *calculer la fréquence de* I_i *dans le Cube C*

$\sup(Y_k, \overline{I_i})=\frac{\text{Count}\ (Y_k \cup \overline{I_i})}{|C|}$; // *calculer la fréquence de* $\overline{I_i}$ *dans le Cube C*

IF $\sup(Y_k, I_i)=0$ or $\sup(Y_k, \overline{I_i})=0$ **then**

j=0 ; // *passer à la partition suivante*

Else

IF ($\sup(Y_k, I_i)\geq$min_sup and $\sup(Y_k, \overline{I_i})\geq$min_sup) **then**

// *Partitionner* p_k *avec* I_i *et ajouter les nouvelles partitions à* $\acute{P}$

$\acute{P}$.add(p_k.Partition_with(I_i)) ;

$\acute{P}$.delete(p_k) ; //*supprimer* p_k *de* $\acute{P}$

j=0 ; // *passer à la partition suivante*

Else // *parcourir les ancêtres des prédicats*

While A is null and $j\geq 1$ **do**

A=Ancestor(i_j,1);

IF A is not null **then**

I_i.Replace(i_j,A) ; //*remplacer* i_j *par son premier ancêtre*

Else

j=j-1 ; // *passer à la partition suivante*

End if

End while

End if

End if

End while

End if

End for

I_i=I'; // *remettre I_i à sa valeur initiale au cas où il a été élargi*

End for

5.5. Partitionnement des données rarement sollicitées

Dans un entrepôt de données contenant des données anciennes ou inactives, comme c'est le cas généralement, ces données n'apparaissent pas dans les requêtes utilisateurs. Ainsi, dans tout algorithme basé sur les requêtes utilisateurs, le partitionnement ne concernera que les données actuellement utilisées, alors que les données non ou rarement sollicitées ne seront pas affectées. Notre algorithme de partitionnement permet quant à lui d'isoler ces données dans une partition séparée que nous appelons « Rest ». Ces données doivent être partitionnées en cas de contrainte de stockage avec un coût de maintenance réduit. Pour traiter ce cas, nous proposons un algorithme complémentaire à l'algorithme de partitionnement principal. La solution proposée consiste à partitionner les données par rang en utilisant la dimension date, jusqu'à satisfaction de la contrainte de taille maximale. L'algorithme commence par identifier le rang de la dimension date approprié pour le partitionnement, selon la taille des données. Par la suite, l'algorithme construit une première partition utilisant le rang identifié. Par exemple, si le rang est l'année, la partition créée P(y) est telle que y=Max(année , Rest)=2018 si 2018 est la plus grande année des données « Rest ». L'algorithme incrémente ensuite la partition P par intervalle de rang jusqu'à atteindre le seuil de taille tolérée (voir Algorithme 3). Cet algorithme de partitionnement par rang peut être utilisé aussi pour fragmenter d'avantage les partitions résultantes de la première phase de partitionnement, si ces dernières ne vérifient pas la contrainte de taille tolérée.

Algorithme 3 : Partitionnement par rang pour les données inactives

Input : *partition Rest*

Partitions: *liste des partitions*

Output Partitions: *liste des partitions*

While $|Rest| > \frac{\max\_size}{Size\,(C)}|C|$ **do** //*Tant que la taille de la partition Rest ne satisfait pas la contrainte de taille max*

i=0 ;

Range=Range_Definition(Rest) ; //*définir l'attribut date à utiliser pour le partitionnement*

Max_Val=Max(Range,Rest) ;//*identifier la valeur maximale, dans Rest, de l'attribut date choisi*

z= [Max_Val] ;

y=[];

While $|P(z)|<\frac{\max\_size}{Size\ (C)}|C|$ **do**

y.add(z[i]) ; // *tant que la taille max n'est pas atteinte on incrémente y selon le rang utilisé, si c'est l'année, on ajoutera les années une par une jusqu'à atteindre la taille max*

i++;

Prev_Period=Pervious_Period(Max_Val,Range,I,Rest); //*Retourne la i ème période précédente*

z.add(Prev_Period);

End while

Partitions.add(P(y)) ; //*ajouter la nouvelle partition*

Rest.delete(P(y)) ; // *supprimer la partition du Rest*

End while

Partitions.update(Rest) ; //*Mettre à jour la nouvelle définition de la partition Rest*

Function Range_Definition(Part partition)

R=[year,semester,quarter,month];// *la liste peut contenir d'autres attributs de la dimension date*

Range=R[0]; // *l'algorithme commence par intervalle d'année*

y=Cube_Max(Range, Part) ; // *Retourne la valeur max du rang dans la partition Part*

int i=0;

While $|P|>\frac{\max\_size}{Size\ (C)}|C|$ and i<3 **do**

i++;

Range=R[i]; // *on passe à un attribut plus petit, le suivant*

y=Cube_Max(Range,Part) ;

End while

Return Range;

5.6. Expérimentation et évaluation

5.6.1. Implémentation

Pour implémenter les algorithmes de notre approche (Letrache et al, 2018), à savoir, l'analyse des requêtes utilisateurs et le partitionnement du cube, nous avons utilisé le langage C#. Ce

dernier permet de manipuler les objets multidimensionnels à travers sa bibliothèque ADOMD.NET (voir Figure 5.5).

En outre, pour tester notre approche, nous avons utilisé la base de données TPC-DS, un support de décision de benchmark (TPC-DS, 2018). Nous avons créé alors le cube OLAP associé, à l'aide de la plateforme Microsoft SQL Server Analysis Services (SSAS). Le cube créé contient une table de fait Store_Sales contenant 24M d'enregistrements et cinq dimensions régulières comme le montre la Figure 5.6 : la dimension Date (73K) contient la hiérarchie date-month-quarter-year, Customer (100K) contient la hiérarchie customer-city-state-country, Item (18K) contient la hiérarchie item-brand-class-category, Store (12) contient elle aussi la hiérarchie store-city-state-country et enfin la dimension Promotion (300). La taille initiale du cube créé est 175.17MB. Enfin, nous avons effectué nos expérimentations sur une machine d'un processeur i3.

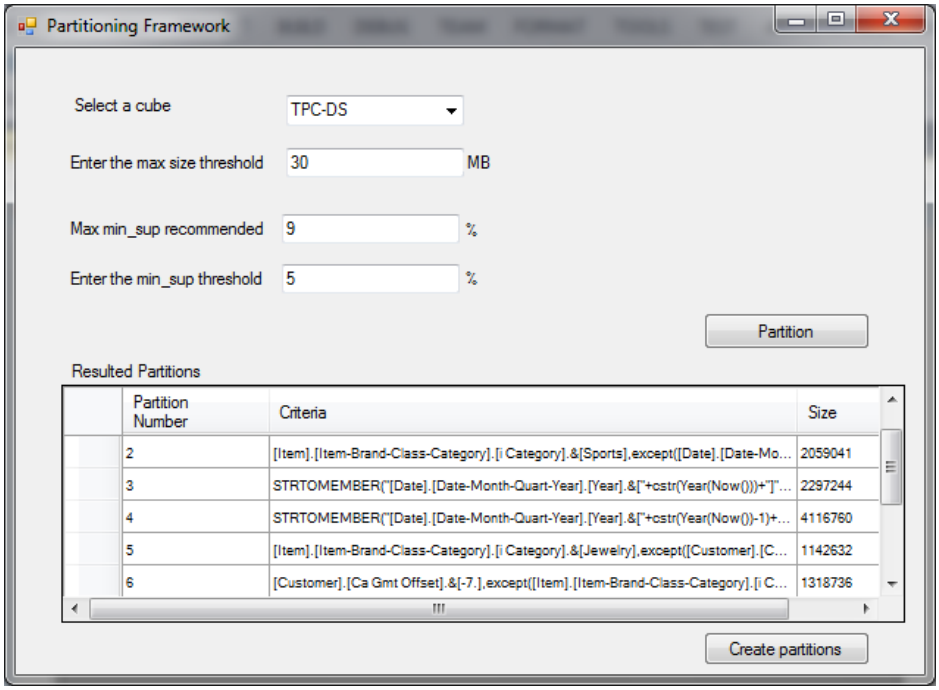


Figure 5.5 Notre Framework de partitionnement OPAR

Nous avons généré par la suite 100 requêtes, en utilisant le générateur de requêtes et les templates TPC-DS, que nous avons converti en langage MDX. Par la suite, nous avons prétraité les requêtes utilisateurs, comme décrit dans la section 5.3, en remplaçant les prédicats de type date par des formules dynamiques et en séparant les sets de tuples et en fixant le facteur de criticité à 1 pour toutes les requêtes.

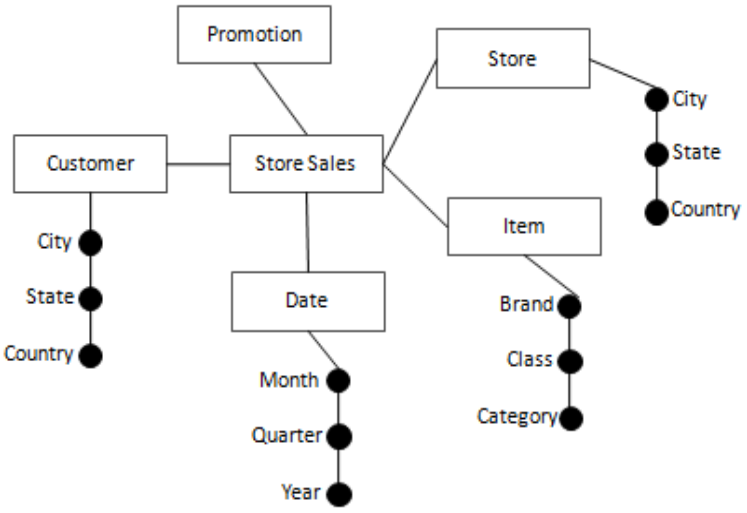


Figure 5.6 Le modèle multidimensionnel TPC-DS utilisé

L'étape qui suit est alors d'exécuter l'algorithme apriori pour déduire les itemsets de prédicats les plus fréquents, et dont nous avons fixé le minimum support à 5%. L'algorithme a identifié ainsi 24 itemsets fréquents. Les résultats montrent que les données les plus sollicitées concernent les deux dernières années, en particulier, les articles relatifs aux catégories sports, livres et bijoux. Par la suite, nous avons passé au partitionnement du cube en utilisant les itemsets fréquents de prédicats. En considérant que la taille tolérée des partitions est 30MB, le min_sup choisi doit être inférieur à 9%. Nous l'avons fixé alors à 5% afin de tester l'aptitude de notre algorithme à élargir la portée des itemsets.

Dans la première itération, l'itemset contient le mois en cours comme seul item, et qui correspond au prédicat le plus utilisé. Cependant, le support de ce dernier est inférieur au seuil min_sup. L'algorithme tente alors d'élargir la portée de l'itemset au lieu de l'ignorer, en remplaçant le prédicat mois par son premier ancêtre dans la hiérarchie date. Le mois est remplacé alors par le trimestre, qui, quant à lui, satisfait la contrainte min_sup. la partition P_1 est alors obtenue. De même, la partition P_5 est obtenue en remplaçant le prédicat « consignment » par son ancêtre, la catégorie de produit « Jewelry ». Nous obtenons enfin les 7 partitions décrites dans le Tableau 5.2.

La partition nommée Rest correspond aux données non-partitionnées, ce sont donc les données pas ou rarement utilisées. Cette partition est alors fragmentée par rang pour respecter la contrainte d'espace de stockage. Dans notre cas, elle est partitionnée par année en trois nouvelles partitions (P_7, P_8, P_9).

Tableau 5.2 Partitions résultantes de notre étude de cas

Partition	Description	Taille (enregistrements)	Taille (MB)
P_1	Trimestre en cours	2040456	12,78
P_2	Année en cours (à l'exception du trimestre en cours et la catégorie sports)	2297244	14,71
P_3	Année précédente	4116760	27,28
P_4	Catégorie sports (à l'exception du trimestre en cours)	2059041	15,19
P_5	Catégorie bijoux (à l'exception des deux dernières années)	1142632	8,24
P_6	Off-7 (à l'exception des deux dernières années)	1318736	7,98
Rest	*Le reste des données non-partitionnées*	*9311584*	*64,44*
P_7	Année en cours – 2	4540120	21,05
P_8	Année en cours – 3	4623029	19,95
P_9	Année en cours – 4	4599136	20,13

5.6.2. Evaluation et Discussion

a. Comparaison avec les résultats de l'état de l'art

Afin de valider notre approche et mesurer son efficacité, nous avons mené une série d'expérimentations dont nous avons comparé les résultats obtenus avec les approches GA, HC, SA (Bellatreche et al, 2009) et EMeD-Part (Toumi et al, 2015). Les paramètres de configuration les plus importants sont décrits dans le Tableau 5.3.

Tableau 5.3 Paramètres de comparaison des approches

Algorithme	Taille de la table de fait	Nombre des requêtes utilisateurs	Nombre d'attributs
GA	24M	60	12
HC	24M	60	12
SA	24M	60	12
EMeD-Part	24M	100	12
Notre approche	24M	100	12

Nous avons commencé par mesurer le temps d'exécution des 100 requêtes utilisateurs, avant et après partitionnement. Nous avons noté alors une grande amélioration des performances, pouvant atteindre les 52% comme le montre la Figure 5.7. Nous avons par la suite mesuré le coût

E/S (entrées/sorties), toujours avant et après partitionnement. Ce coût correspond au nombre de pages nécessaires à charger le cube ou les partitions dans la mémoire afin de répondre à une requête spécifique, tel que exprimé par la formule suivante :

Coût(Q_j)=$\sum_{i=0}^{n} \frac{size\,(P_i)}{PS}$ où n est le nombre de partitions requises pour répondre à une requête utilisateurs Q_j et PS est la taille d'une page système.

La Figure 5.8 affiche le pourcentage de réduction du coût E/S calculé par rapport au cube non partitionné. Le résultat obtenu représente 11,4% comparé au coût E/S du cube non partitionné, ce qui représente un résultat très encourageant comparé aux autres approches (voir Tableau 5.4).

En outre, nous avons mené plusieurs expérimentations pour mesurer la performance de notre algorithme. Nous avons donc mesuré le nombre d'itérations de l'algorithme pour différentes valeurs de min_sup comme le montre la Figure 5.9. Ainsi, le nombre d'itérations de notre algorithme reste inférieur à 500 indépendamment de la valeur du min_sup, au moment où il est de l'ordre de milliers (Toumi et al, 2015) dans les autres approches (2500 dans le meilleur des cas). Par conséquent, notre algorithme nécessite 5 fois moins d'itérations que l'approche EMeD et les approches heuristiques.

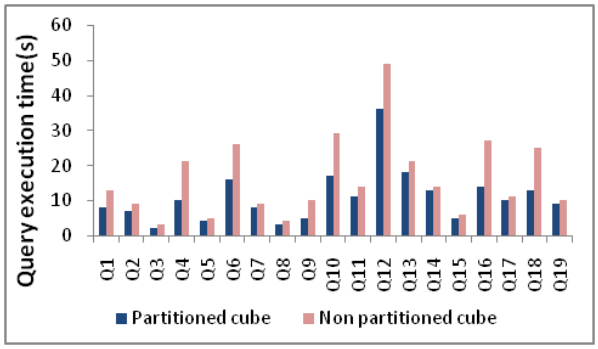


Figure 5.7 Temps d'exécution des requêtes

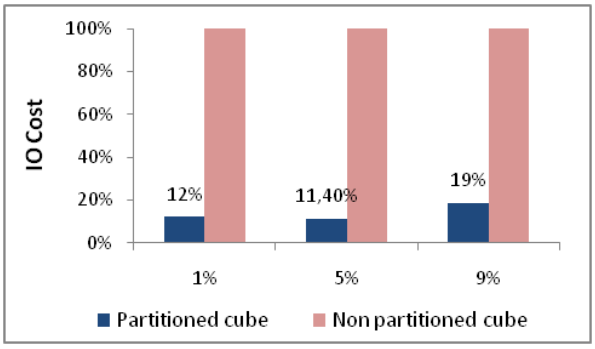


Figure 5.8 Coût E/S des requêtes

Nous avons mesuré également le nombre de partitions générées par rapport au seuil min_sup (voir Figure 5.10). Nous avons constaté que le nombre de partitions obtenues reste toujours raisonnable, ce qui se traduit par un coût de maintenance réduit.

Par exemple, le nombre de partitions produites par notre algorithme pour le cas étudié est 9 partitions au moment où l'algorithme génétique produit 80 partitions (Bellatreche et al, 2009). Le Tableau 5.4 résume les résultats de comparaison de notre approche avec celles existantes pour le même volume de données et charge de travail et qui ne requièrent pas une configuration spécifique.

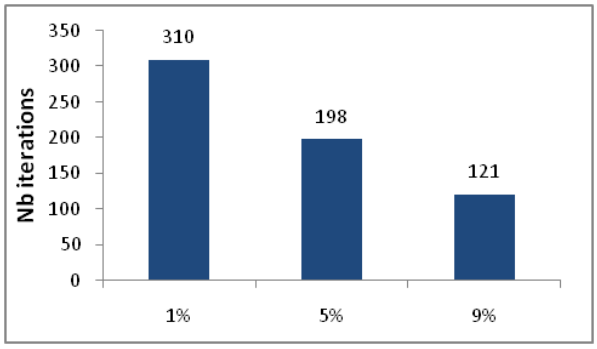


Figure 5.9 Nombre d'itérations vs min_sup

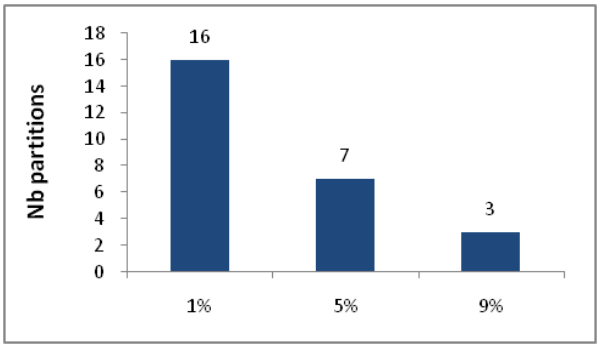


Figure 5.10 Nombre de partitions vs min_sup

En plus de l'amélioration des requêtes utilisateurs, notre approche permet de générer des partitions dynamiques par rapport à la dimension date, ce qui n'est pas supporté par les autres approches puisqu'elles traitent les prédicats date comme des valeurs statiques. De plus, pour diminuer le nombre de partitions et ainsi le coût de maintenance, notre approche intègre une stratégie complémentaire de partitionnement par rang pour les données rarement sollicitées (Rest) ou pour ajuster la taille des partitions à un certain seuil. Par contre, les approches existantes partitionnent la totalité du cube selon les prédicats des requêtes utilisateurs, ce qui peut entrainer la génération d'un grand nombre de partitions. De plus, dans les approches basées sur un modèle de coût (Bellatreche et al, 2009 ; Toumi et al, 2015), l'administrateur BI doit définir des paramètres compliqués comme le nombre maximal de partitions, ce qui n'est pas le cas pour notre approche.

Tableau 5.4 Evaluation de notre approche

Algorithme	Moyenne d'amélioration du temps d'exécution des requêtes	Coût E/S vs. Cube non-partitionné	Nombre d'itérations	Nombre de partitions résultantes
GA	13%	37.5%	≅5500	80
HC	10.07%	37.78%	-	96
SA	26.72%	38.13%	-	80
EMeD_Part	-	12.5%	≅2500	8
Notre approche	28.1%	11.04%	198	9

D'autre part, le partitionnement des cubes OLAP améliore également le temps de rafraîchissement, pouvant varier de quelques secondes à plusieurs heures selon la taille de la partition, comme le montre la Figure 5.11. Par ailleurs, la réduction du temps de traitement peut

aider dans la définition d'une stratégie appropriée de mise à jour du cube. Par exemple, dans notre étude de cas (voir Figure 5.12), la partition P_1 relative aux données du trimestre en cours et dont la durée de traitement est d'environ 10 seconds, peut être utilisée pour les reporting journaliers mais aussi pour rétablir rapidement ceux urgents en cas de problème de synchronisation. Elle peut ainsi être traitée plusieurs fois par jour si nécessaire ou uniquement une fois par soir, alors que les partitions P_2et P_3 peuvent être traitées une fois par mois et P_4, P_5, P_6, P_7, P_8 et P_9 une fois par année.

Enfin, nous avons mené des expérimentations supplémentaires pour mesurer la sensibilité de notre approche. En effet, étant donné que notre algorithme de partitionnement est basé sur la charge de travail, cela signifie que tant que les requêtes utilisateurs ne changent pas, alors les performances du cube restent bonnes. En fait, le changement dans les requêtes peut concerner les colonnes de projection ou celles de sélection. Pour expérimenter le premier cas, nous avons considéré un template de requête Q' dont nous avons modifié les colonnes de projection. Nous avons noté alors que l'amélioration des performances reste au dessus de 28%, comme le montre la Figure 5.13. Nous avons ensuite modifié les prédicats de la requête (colonnes de sélection). Dans ce cas, nous avons constaté une régression des performances, comme le montre la Figure 5.14, variant de 3% à 28%.

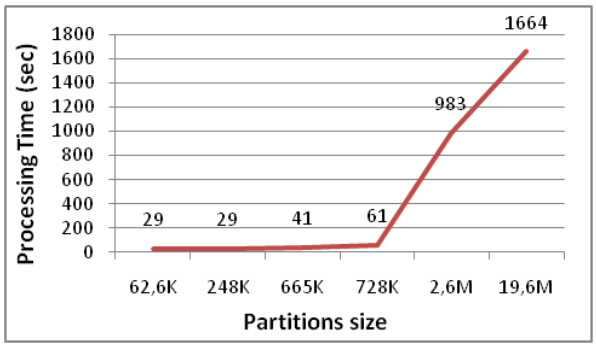


Figure 5.11 Temps de traitement vs. taille des partitions

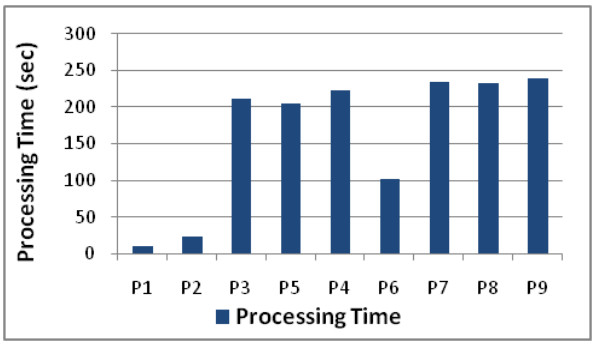


Figure 5.12 Temps de traitement des partitions de notre étude cas

Pour cette raison les performances du cube, de requêtage et de traitement, doivent être vérifiées périodiquement pour identifier toute régression des performances, en raison par exemple d'un changement des perspectives business ou d'augmentation exceptionnelle du volume de données, et ainsi ré-exécuter l'algorithme de partitionnement.

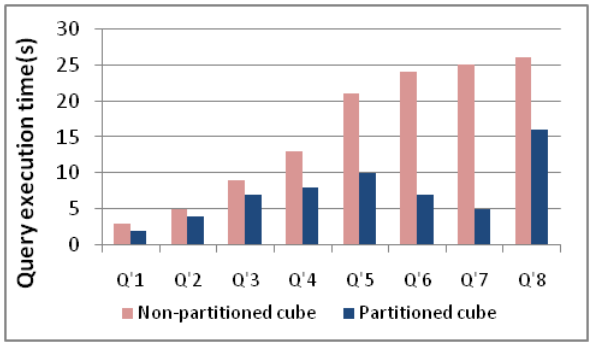


Figure 5.13 Sensibilité vs changement des colonnes de projection

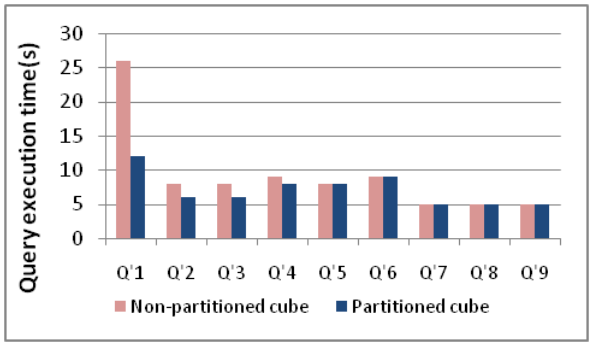


Figure 5.14 Sensibilité vs changement des colonnes de sélection

b. Comparaison avec l'algorithme génétique

La deuxième expérimentation que nous avons menée pour valider notre approche est de la comparer avec celle génétique (GA), représentant l'approche la plus utilisée dans la littérature comme approche de base ou de comparaison.

Comme présenté dans le chapitre précédent (section 4.4.3), le principe de l'approche génétique est de représenter chaque schéma de partitionnement sous forme d'un chromosome dont les gènes représentent les attributs de partitionnement. Les chromosomes subissent un ensemble d'opérations génétiques (sélection, mutation, croisement) pour donner de nouveaux chromosomes. Enfin, pour sélectionner la meilleure solution (chromosome), on utilise un modèle de coût calculant le nombre d'entrées/sorties nécessaires pour répondre à chaque requête utilisateurs (voir Algorithme 4).

Algorithme 4 : Algorithme génétique pour le partitionnement

Require: SD : *liste des domaines de sélection des attributs*
Q : *liste des requêtes fréquentes*
B : *Nombre maximal de partitions*
Ensure: chromosome //*retourner le chromosome ayant le plus bas coût d'E/S*
//Générer aléatoirement une population initiale
For (i = 0; i < 40; i + +) **do**
chromosome=Create_chromosome();
For all (s in SD) **do**
a=CreateRandomArray(s.lenght);
chromosome.genes.add(a);
End for
population.add(chromosome);

```
End for
//Exécution des opérations GA
Select.SetType(Elitism)
SelectFitest(population, FitnessFunction);
While (nb_generation <= 1000) do
        Crossover(population,0.7,DoublePoint) ;
        Mutation(population,0.1) ;
        SelectFitest(population, FitnessFunction);
End while
chromosome=FitestChromosome(population);//sélectionner le chromosome ayant la meilleure fitness correspondant au plus bas coût d'E/S
//Fonction de Fitness
FitnessFunction(Chromosome c)
        Fragments=ChromosomeToFragments(c);
        For all (query in Q) do
                For all (F in Fragments) do
                        Cost+ = CalculateQueryCost(q, F); //calculer le coût de chaque fragment pour chaque requête
                End for
        End for
        IF (Fragments.count/B >1)
                Cost*= Fragments.count/B ; //multiplier le coût par la pénalité si le nombre de partitions générées dépasse le seuil B
        Return Cost
```

Pour implémenter ce dernier, nous avons utilisé le langage C# via son Framework GAF (Genetic Algorithm Framwork) (GAF, 2016). Nous avons utilisé le même schéma de base données TPC-DS tel que présenté dans la section précédente. Nous avons par la suite identifié le domaine de sélection de chaque attribut ainsi que les facteurs de sélectivité des prédicats.

L'algorithme commence alors par créer une première population que nous avons fixée à 40 chromosomes, telle que dans (Bellatreche et al, 2005). Les chromosomes sont créés en construisant aléatoirement leurs gènes, associés chacun à un attribut de partitionnement et représenté par un tableau d'entiers de même longueur que le domaine de sélection associé à cet attribut. L'opération de croisement est paramétrée à 0.7 et double point, et la mutation est fixée à

0.1% (Bellatreche et al, 2005). Dans notre étude de cas, le nombre maximal de partitions résultantes est 768, ce qui représente un grand nombre de partitions à créer et à gérer. Nous avons ajouté alors un seuil B sur le nombre de partitions générées, que nous avons fixé à 15. Nous avons aussi modifié la fonction de fitness de façon à intégrer une fonction de pénalité afin d'éliminer les solutions produisant un grand nombre de partitions. La fonction de fitness retenue est:

Cost'(Q)=Cost(Q) x $\frac{N}{B}$ où N est le nombre de partitions résultantes. Enfin, le nombre de partitions obtenues de l'algorithme génétique est 8.

Nous avons alors mené plusieurs expérimentations pour comparer notre approche et celle génétique commençant par comparer le temps d'exécution des requêtes. Ainsi, nous avons noté que l'approche OPAR fournit un temps d'exécution meilleur que celui de l'approche GA de 26% en moyenne (voir Figure 5.15).

Par la suite, nous avons comparé le coût d'E/S en utilisant la même formule utilisée dans la section précédente : Cost(Q)=$\sum_{i=1}^{n} \frac{size(P_i)}{Ps}$ où n est le nombre de partitions nécessaires pour répondre à une requête Q et PS est la taille d'une page système. La comparaison montre que le coût d'E/S de l'approche OPAR est meilleur que celui de l'approche GA d'environ 51% (voir Figure 5.16). Ceci peut être expliqué par le fait que l'approche OPAR améliore toutes les requêtes vu qu'elle se base sur les itemsets fréquents de prédicats, alors que l'approche GA améliore les requêtes fréquentes uniquement. De plus, l'approche GA, à cause du produit cartésien des prédicats, génère des solutions inexploitables pour les données qui ne sont pas fréquemment sollicitées. Au moment où l'approche OPAR isole ces données dans une partition qui peut être gérée séparément.

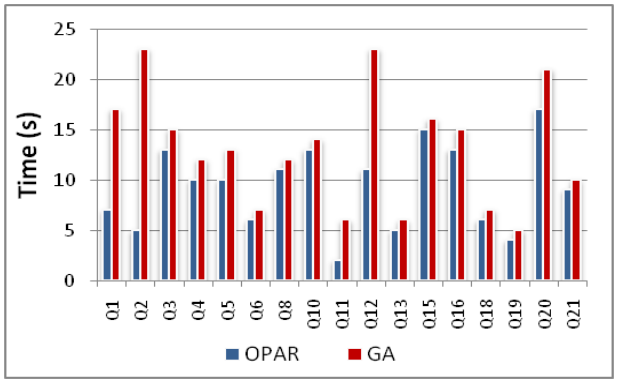


Figure 5.15 Temps d'exécution des approches OPAR et GA

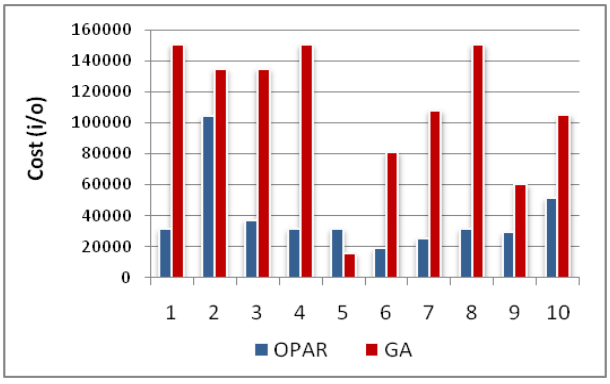


Figure 5.16 Coût E/S des approches OPAR et GA

Enfin, nous avons comparé la taille des partitions résultantes pour chacune des approches. Nous avons noté alors que l'approche OPAR génère des partitions de tailles régulières (entre 8 et

27MB) comme le montre la Figure 5.17, grâce aux paramètres min_sup et max_size, permettant de contrôler la taille des partitions. Par contre, l'approche GA, qui contrôle uniquement le nombre de partitions, génère des partitions de tailles irrégulières (entre 0.08 et 83MB) comme le montre la Figure 5.18.

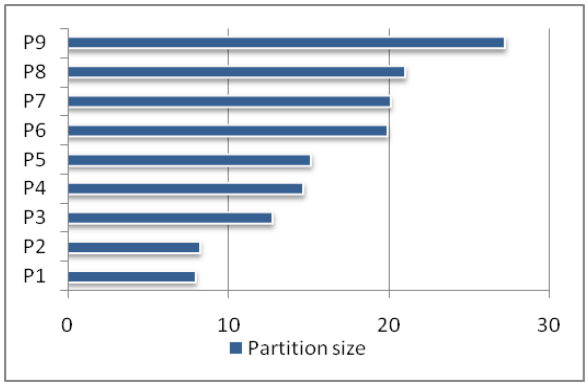


Figure 5.17 Taille des partitions résultantes de l'approche OPAR

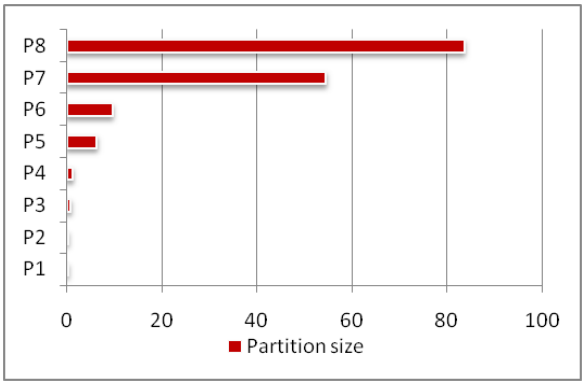


Figure 5.18 Taille des partitions résultantes de l'approche GA

5.7. Conclusion

Dans ce chapitre, nous avons présenté notre approche OPAR de partitionnement des cubes OLAP, visant à améliorer le temps de réponse des requêtes utilisateurs et le temps de traitement du cube. Elle vise aussi à faciliter la maintenance des bases OLAP et la gestion de l'espace de stockage. En effet, contrairement aux approches précédentes, qui se sont focalisées uniquement sur les requêtes utilisateurs, l'approche OPAR prend en considération, en plus de la fréquence des requêtes utilisateurs, la criticité de ces dernières et le volume de données. L'approche OPAR commence alors par analyser les requêtes utilisateurs pour en déduire les itemsets fréquents de prédicats. Par la suite, en se basant sur ces itemsets, l'algorithme de partitionnement identifie la stratégie de partitionnement appropriée selon les seuils, support et taille, prédéfinis. Ceci permet de garantir la régularité des tailles des partitions résultantes, et ainsi, faciliter la gestion de l'espace de stockage. Il permet aussi d'améliorer le temps de traitement du cube et sa gestion, dépendants eux aussi de la taille des partitions. La phase d'analyse des requêtes utilisateurs inclut une phase de prépararion visant principalement à intégrer un facteur de criticité pour les requêtes critiques mais non fréquentes. Cette phase concerne également le traitement des prédicats de type date pour permettre par la suite un partitionnement dynamique par rapport à la dimension date.

En outre, l'approche OPAR, dans le but de fournir les meilleures performances, tente de créer des partitions qui s'adaptent aux requêtes, ou du moins, les englobent en se basant sur le concept de hiérarchie.

Outre cela, l'approche OPAR, contrairement aux autres approches, permet d'isoler les données rarement utilisées dans une partition séparée qui peut être partitionnée selon la contrainte de stockage, et ainsi réduire le coût de maintenance, au lieu d'utiliser la même stratégie de partitionnement que celle des données fréquemment sollicitées.

Notre proposition comprend ainsi les trois avantages principaux du partitionnement à savoir : l'amélioration des performances, la maintenance (mise à jour du cube) et la gestion de stockage.

Enfin, les résultats des expérimentations affirment l'efficacité de notre approche et l'amélioration considérable des performances après partitionnement.

Conclusion et perspectives

Bilan

Ce mémoire présente nos contributions s'inscrivant dans le cadre de la méta-modélisation, le développement et l'optimisation des systèmes décisionnels, en particulier des cubes OLAP.

En fait, malgré le nombre de travaux qui ont été menés dans le cadre de la modélisation et l'élaboration des systèmes décisionnels, on constate que plusieurs aspects du projet sont toujours sujets de débats et de recherche que ce soit au niveau de la démarche de développement qu'au niveau de la conception et de l'architecture.

Ainsi, pour mettre en place un système décisionnel, la première préoccupation des concepteurs BI, est le choix d'un type de démarche parmi les trois types possibles: orientée données, orientée besoins utilisateurs et orientée buts. Le premier type s'avère difficile à adopter, vu le nombre et la taille des systèmes opérationnels des organisations ainsi que le nombre d'informations qu'ils contiennent. Ce type de démarche peut engendrer des développements coûteux et inutiles. Les approches orientées utilisateurs, quant à elles, embrassent les besoins des utilisateurs et s'avèrent plus réalistes. Le concepteur BI doit, toutefois, gérer les déviations des besoins des utilisateurs pour ne pas engendrer, une fois encore, plusieurs développements. Les approches orientées buts permettent de remédier à cela, en se focalisant sur les besoins stratégiques des décideurs, visant à unifier la vision de l'entreprise. Il reste alors à réconcilier ces buts avec les sources de données pour juger la faisabilité du développement et définir la source de chaque information.

La deuxième préoccupation est alors de modéliser ces besoins, exprimés sous forme de buts. A cet effet, plusieurs techniques ont été proposées dans la littérature comme vu dans le chapitre 1, mais qui présentent cependant plusieurs inconvénients notamment la complexité.

Une fois les besoins exprimés, on passe ensuite à la modélisation du système puis à l'implémentation qui est étroitement liée aux plateformes utilisées étant donné qu'elles offrent chacune un environnement différent.
L'implémentation comprend plusieurs étapes dont la création des tables au niveau du DW et celles de chargement, la création des cubes OLAP, dimensions, hiérarchies, en plus de l'implémentation des ETLs de chargement etc. Des tâches qui se répètent pour chaque nouveau data mart à mettre en place, ou nouvelle métrique ou axe d'analyse à intégrer. D'où l'intérêt d'automatiser un tel processus itératif. Pour ce faire, plusieurs techniques ont été proposées dans la littérature, notamment des développements personnalisés. Ces solutions reposent, cependant, sur des langages limités et non dédiés et qui requièrent donc des développements compliqués.

Parmi ces solutions d'automatisation, on trouve la MDA. Cette dernière offre plusieurs avantages. Tout d'abord, la MDA se base sur une architecture à quatre niveaux visant à séparer la couche métier et modélisation de plateforme utilisée (relationnel, OLAP, objet etc.) garantissant ainsi la pérennité des modèles. De plus, grâce à ses concepts de base, à savoir l'abstraction, la méta-modélisation et les transformations automatiques, la MDA permet, d'une part, la portabilité et l'interopérabilité des applications et d'une autre part, l'automatisation du processus de développement. Ainsi, un seul développement est valable pour tout nouveau besoin. La MDA propose aussi des langages de transformation offrant une grande flexibilité dans la définition des règles de transformation. De cette manière, pour migrer d'une plateforme vers une autre, il suffit de définir les règles de transformation relatives à cette nouvelle plateforme.

Etant donné tous ses avantages, l'architecture MDA a été largement utilisée dans les travaux de l'état de l'art. Cependant, comme discuté dans le chapitre 2, toutes les approches proposées se sont focalisées sur le modèle relationnel alors que le modèle OLAP a été négligé.

Notre contribution est, alors, d'introduire une approche MDA orientée but pour l'élaboration des systèmes décisionnels allant des spécifications des besoins jusqu'à l'implémentation finale.

La première phase de notre approche est la modélisation des buts stratégiques des utilisateurs. Pour cet effet, nous avons proposé deux profils : le GoalCases qui étend le diagramme de cas d'utilisation et qui sert à modéliser les buts stratégiques des décideurs, et le SGAP, un profil qui étend le modèle BPMN, et qui sert à détailler les besoins modélisés dans le GoalCases. Dans ce modèle SGAP, nous décomposons les buts en sous buts puis en informations, jusqu'à définir les mesures et dimensions qui vont constituer, par la suite, le schéma multidimensionnel. Le SGAP intègre aussi la phase de réconciliation des besoins avec les sources de données. Par la suite,

nous déduisons, à partir de ces spécifications un modèle multidimensionnel indépendant de la plateforme utilisée, conformément à l'architecture MDA (modèle PIM). Nous proposons pour cette fin un métamodèle UML permettant de modéliser les concepts multidimensionnels, en particulier le type de relation plusieurs-à-plusieurs. Par la suite, deux modèles spécifiques aux plateformes utilisées sont alors obtenus automatiquement à partir du modèle PIM, à savoir, le modèle relationnel et le modèle OLAP. Pour les modéliser, nous avons opté pour le standard OIM (Open Information Model), un choix que nous avons justifié dans le chapitre 2. A ce niveau, nous avons proposé également un modèle PDM servant à décrire la plateforme OLAP cible afin de permettre le passage automatique du modèle PIM vers celui OLAP pour n'importe quelle plateforme.

La dernière étape de notre approche MDA est la génération automatique du code d'implémentation du DW et des cubes OLAP à partir des deux modèles spécifiques obtenus précédemment, et ce via des règles de transformation de type modèle-à-texte. Ainsi, en plus du code SQL de création du DW, nous avons généré le code XMLA d'implémentation des cubes OLAP relatif à la plateforme SSAS. Enfin, nous avons validé notre approche via une étude de cas réel.

Notre seconde contribution présentée dans ce mémoire se rapporte à l'optimisation des systèmes décisionnels. En effet, vu le volume colossal des données qu'ils stockent, les systèmes décisionnels peuvent connaître des dégradations importantes des performances. Ces dernières concernent principalement le temps de réponse des requêtes et le temps de rafraîchissement des cubes OLAP. En outre, à cause de la croissance permanente de leurs tailles, les DSS engendrent des problèmes de gestion d'espace de stockage.

Pour pallier à cela, plusieurs techniques d'optimisation existent, dont le partitionnement horizontal est la technique la plus recommandée. Le défi est alors de définir la stratégie de partitionnement appropriée à chaque DSS.

Dans la littérature, plusieurs travaux ont été menés dans ce sens. La plupart se sont focalisés sur les bases de données et entrepôts relationnels et sont d'une grande complexité en terme de nombre de partitions générées.

Ainsi, notre contribution est une approche de partitionnement horizontal des cubes OLAP que nous avons appelé OPAR. Son objectif est d'améliorer les performances des cubes OLAP et réduire leur coût de maintenance. Basée sur l'algorithme des règles d'association, l'approche OPAR prend en compte, à la fois, le besoin des utilisateurs à travers la fréquence et la criticité de leurs requêtes, ainsi que la régularité des tailles des partitions résultantes.

Pour ce faire, l'approche OPAR comporte deux phases principales. La première concerne l'analyse des requêtes utilisateurs pour déduire les itemsets fréquents de prédicats. Cette phase inclut également une étape de préparation des requêtes, en particulier le traitement des prédicats de type date et des opérateurs logiques ainsi que l'intégration du facteur de criticité.

Par la suite, l'algorithme de partitionnement proposé fragmente le cube en prenant en entrée les itemsets résultants de la première phase. L'algorithme tente de créer des partitions respectant un seuil min_sup et dont la définition dépend du seuil de taille des partitions autorisé par l'administrateur du système. L'algorithme proposé se base sur le concept de hiérarchie pour élargir la population des itemsets fréquents qui ne satisfont pas la contrainte min_sup.

Pour valider l'approche OPAR, nous avons mené plusieurs expérimentations, sur une base de benchmark, et dont les résultats étaient dans la majorité favorables, comparés aux résultats de l'état de l'art.

Nous avons également implémenté l'algorithme génétique, l'un des algorithmes les plus utilisés dans la littérature et que nous avons comparé à notre algorithme sur la même base de bechmark TPC-DS.

En résumé, l'approche OPAR offre une solution d'optimisation qui s'adapte aux besoins des utilisateurs et qui permet également de réduire le coût de maintenance en facilitant la gestion de l'espace ainsi que la définition de stratégies de rafraîchissement adaptées aux besoins des utilisateurs et administrateurs.

Perspectives

Plusieurs travaux de recherche sont envisagés pour améliorer d'avantage les solutions proposées dans le cadre de la modélisation et l'automatisation du cycle de développement des systèmes décisionnels ainsi que l'optimisation de ces derniers. Ainsi, nos perspectives sont :

- Au niveau de la modélisation, plusieurs aspects restent à traiter notamment la réconciliation automatique des besoins utilisateurs avec les sources de données, l'une des difficultés majeures dans le développement des projets décisionnels. Un autre volet à traiter est celui de l'automatisation des outils de chargement des données, toujours dans le cadre d'une architecture MDA. Notre ambition est, en fait, d'automatiser tous les aspects d'élaboration des DSS pour ainsi permettre aux développeurs de se focaliser sur le cœur du métier au lieu des outils mais aussi de réduire le coût, délai et ressources requises pour le développement et la maintenance du projet.
- Un autre aspect important à traiter également, est celui de la mise à jour des métadonnées après leur intégration automatique. En effet, notre approche permet de régénérer les

métadonnées en cas de modification au niveau des modèles CIM, PIM ou PSM. Le souci est, cependant, de conserver les modifications manuelles faites par l'administrateur au niveau de l'entrepôt des métadonnées.

- Pour compléter l'architecture OMDA, nous envisageons d'intégrer le processus ETL également.
- Dans le cadre de l'optimisation et afin d'améliorer d'avantage les résultats de l'approche OPAR, nous envisageons, en premier lieu de tester la combinaison de plusieurs méthodes d'optimisation. Ainsi, nos travaux concerneront d'abord l'optimisation du DW et du cube OLAP à la fois, pour améliorer le temps de rafraîchissement du DW et des cubes OLAP. Ensuite, combiner plusieurs méthodes d'optimisation.
- Toujours dans le cadre de l'optimisation, nous envisageons de traiter l'optimisation des processus de chargement ETL.
- Enfin, nous envisageons d'étudier l'applicabilité de nos deux approches OMDA et OPAR dans le contexte big data.

Références

Agrawal, R., Imieliński, T., & Swami, A. (1993, June). Mining association rules between sets of items in large databases. In Acm sigmod record (Vol. 22, No. 2, pp. 207-216). ACM.

AlHammad, N., & Taha, Y. (2016, November). Performance evaluation study of data retrieval in data warehouse environment. In Proceedings of the 2nd International Conference on Communication and Information Processing (pp. 48-54). ACM.

Alshboul, R. (2012). Data warehouse explorative study. Applied Mathematical Sciences, 6(61), 3015-3024.

Aouiche, K., Darmont, J., Boussaïd, O., & Bentayeb, F. (2005, August). Automatic selection of bitmap join indexes in data warehouses. In International Conference on Data Warehousing and Knowledge Discovery (pp. 64-73). Springer, Berlin, Heidelberg.

Apache-Jmeter Users Manual. 2004. https://jmeter.apache.org/. accessed 22/06/2018

Atigui, F., Ravat, F., Tournier, R., & Zurfluh, G. (2011). A Unified Model Driven Methodology for Data Warehouses and ETL Design. In ICEIS (1) (pp. 247-252).

ATL: Atlas transformation language, ATL User Manual. Version 0.7. February 2006.

Azzini, A., Marrara, S., Maurino, A., & Topalovic, A. (2017). MMBR: A Report-driven Approach for the Design of Multidimensional Models. In SIMPDA (pp. 83-97).

Bellatreche, L. (2003). Techniques d'optimisation des requêtes dans les data warehouses. In 6th International Symposium on Programming and Systems (ISPS 03), Alger, Algérie (pp. 81-98).

Bellatreche, L., & Boukhalfa, K. (2005, August). An evolutionary approach to schema partitioning selection in a data warehouse. In International Conference on Data Warehousing and Knowledge Discovery (pp. 115-125). Springer, Berlin, Heidelberg.

Bellatreche, L., Boukhalfa, K., Richard, P., & Woameno, K. Y. (2009). Referential horizontal partitioning selection problem in data warehouses: Hardness study and selection algorithms. International Journal of Data Warehousing and Mining (IJDWM), 5(4), 1-23.

Bellatreche, L., Karlapalem, K., Mohania, M., & Schneider, M. (2000). What can partitioning do for your data warehouses and data marts?. In Database Engineering and Applications Symposium, 2000 International (pp. 437-445). IEEE.

Bézivin, J., & Blanc, X. (2002). MDA: Vers un important changement de paradigme en génie logiciel. Développeur référence v2, 16, 15.

Bézivin, J., Dupé, G., Jouault, F., Pitette, G., & Rougui, J. E. (2003, October). First experiments with the ATL model transformation language: Transforming XSLT into XQuery. In 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture (Vol. 37).

Bézivin, J., & Briot, J. P. (2004). Sur les principes de base de l'ingénierie des modèles. L'OBJET, 10(4), 145-157.

Blanco, C., Fernández-Medina, E., & Trujillo, J. (2015a). An MDA approach for developing Secure OLAP applications: metamodels and transformations. Computer Science and Information Systems, 12(2), 541-565.

Blanco, C., de Guzmán, I. G. R., Fernández-Medina, E., & Trujillo, J. (2015b). An architecture for automatically developing secure OLAP applications from models. Information and Software Technology, 59, 1-16.

Cephas Consulting Corp (2006) The Fast Guide to Model Driven Architecture The Basics of Model Driven Architecture (Whitepaper). http://www.omg.org/mda/mda_files/Cephas_MDA_Fast_Guide.pdf

Ceri, S., Negri, M., & Pelagatti, G. (1982, June). Horizontal data partitioning in database design. In Proceedings of the 1982 ACM SIGMOD international conference on Management of data (pp. 128-136). ACM.

Cheung D.W. Zhou B, Kao B, Kan H, Lee SD (2001) Towards the building of a dense-region-based OLAP system, Data and Knowledge Engineering 36,pp 127

CWM: Common warehouse metamodel (CWM) specification version 1.1, volume 1. March 2003.

Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. IBM Systems Journal, 45(3), 621-645.

Dataconomy: THE 20 MOST POPULAR BUSINESS INTELLIGENCE TOOLS (2017). http://dataconomy.com/2017/02/top-20-bi-tools. 20/02/2017

El Akkaoui, Z., Mazón, J. N., Vaisman, A., & Zimányi, E. (2012, September). BPMN-based conceptual modeling of ETL processes. In International Conference on Data Warehousing and Knowledge Discovery (pp. 1-14). Springer, Berlin, Heidelberg.

El Beggar, O., Letrache, K., & Ramdani, M. (2016, October). Towards an MDA-oriented UML profiles for data warehouses design and development. In Intelligent Systems: Theories and Applications (SITA), 2016 11th International Conference on (pp. 1-6). (IEEE).

El Beggar, O., Letrache, K., & Ramdani, M. (2017). CIM for data warehouse requirements using an UML profile. IET Software, 11(4), 181-194.

GAF : The Genetic Algorithm Framework for .Net. (December 2016) https://gaframework.org/wiki/index.php/The_Genetic_Algorithm_Framework_for_.Net

Gaševic, D., Djuric, D., & Devedžic, V. (2006). Model driven architecture and ontology development. Springer Science & Business Media.

Giorgini, P., Rizzi, S., & Garzetti, M. (2008). GRAnD: A goal-oriented approach to requirement analysis in data warehouses. Decision Support Systems, 45(1), 4-21.

Golfarelli, M., Maio, D., & Rizzi, S. (1998). The dimensional fact model: A conceptual model for data warehouses. International Journal of Cooperative Information Systems, 7(02n03), 215-247.

Gorla, N. (2003). Features to consider in a data warehousing system. Communications of the ACM, 46(11), 111-115.

Han, J., Pei, J., & Kamber, M. (2011). Data mining: concepts and techniques. Elsevier.

Hayen, R. L., Rutashobya, C. D., & Vetter, D. E. (2007). An investigation of the factors affecting data warehousing success. International Association for Computer Information Systems (IACIS), 8(2), 547-553.

Hughes, R. (2012). Agile data warehousing project management: business intelligence systems using Scrum. Newnes.

Informatica. White paper (2013) Metadata Management for Holistic Data Governance-Using Informatica Metadata Manager & Business Glossary to Govern Your Data and Deliver Better Business Decisions

Inmon, W. H. (2005). Building the data warehouse. John wiley & sons.

Johnson, M. (2011) Data Governance: What You Need to Know for IT Operations Management, Lightning Source.

Jouault, F., & Kurtev, I. (2006, April). On the Architectural Alignment of ATL and QVT. In Proceedings of the 2006 ACM symposium on Applied computing (pp. 1188-1195). ACM

Jovanovic, P., Romero, O., Simitsis, A., Abelló, A., & Mayorova, D. (2014). A requirement-driven approach to the design and evolution of data warehouses. Information Systems, 44, 94-119.

Khatri, V., & Brown, C. V. (2010). Designing data governance. Communications of the ACM, 53(1), 148-152.

Kimball, R., & Ross, M. (2011). The data warehouse toolkit: the complete guide to dimensional modeling. John Wiley & Sons.

Kimball, R., & Ross, M. (2013). The data warehouse toolkit: The definitive guide to dimensional modeling. John Wiley & Sons.

Kleppe, A. G., Warmer, J., Warmer, J. B., & Bast, W. (2003). MDA explained: the model driven architecture: practice and promise. Addison-Wesley Professional.

Letrache, K., El Beggar, O., & Ramdani, M. (2017). The automatic creation of OLAP cube using an MDA approach. Software: Practice and Experience, 47(12), 1887-1903. John Wiley & Sons.

Letrache, K., El Beggar, O., & Ramdani, M. (2018). OLAP cube partitioning based on association rules method. Applied Intelligence, 1-15. Springer

Letrache K, El Beggar O, Ramdani, M (2018) Comparative analysis of our association rules based approach and a genetic approach for OLAP partitioning. International Conference on Networked Systems(Netys 2018). Lecture Notes in Computer Science. Springer

Letrache, K., El Beggar, O., & Ramdani, M. (2016, October). Modeling and creating KPIs in MDA approach. In Information Science and Technology (CiSt), 2016 4th IEEE International Colloquium on (pp. 222-227). IEEE.

Luján-Mora, S., Trujillo, J., & Song, I. Y. (2006). A UML profile for multidimensional modeling in data warehouses. Data & Knowledge Engineering, 59(3), 725-769.

Mahboubi, H., & Darmont, J. (2008, October). Data mining-based fragmentation of XML data warehouses. In Proceedings of the ACM 11th international workshop on Data warehousing and OLAP (pp. 9-16). ACM.

Mami, I., & Bellahsene, Z. (2012). A survey of view selection methods. ACM SIGMOD Record, 41(1), 20-29.

Marco, D., Jennings, M. (2004) Universal Meta Data Models.John Willey

Maté, A., Trujillo, J., & Franch, X. (2014). Adding semantic modules to improve goal-oriented analysis of data warehouses using I-star. Journal of systems and software, 88, 102-111.

Mazón, J. N., Pardillo, J., & Trujillo, J. (2007, November). A model-driven goal-oriented requirement engineering approach for data warehouses. In International Conference on Conceptual Modeling (pp. 255-264). Springer, Berlin, Heidelberg.

Mazón, J. N., & Trujillo, J. (2008). An MDA approach for the development of data warehouses. Decision Support Systems, 45(1), 41-58.

Mazón, J. N., & Trujillo, J. (2009). Data Warehousing Meets MDA. In New Trends in Data Warehousing and Data Analysis(pp. 1-20). Springer, Boston, MA.

MDC-OIM : Meta Data Coalition Open Information Model Version 1.1 (August, 1999)

Messaoud, R. B., Rabaséda, S. L., Boussaid, O., & Missaoui, R. (2006, November). Enhanced mining of association rules from data cubes. In Proceedings of the 9th ACM international workshop on Data warehousing and OLAP (pp. 11-18). ACM.

MS-SSP: SQL Server Profiler. (October 24, 2016) https://docs.microsoft.com/en-us/sql/tools/sql-server-profiler/sql-server-profiler?view=sql-server-2017

Miller, J., Mukerji, J. (July 9, 2001) Model Driven Architecture (MDA) Document number ormsc/2001-07-01. Architecture Board ORMSC. https://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

Muñoz, L., Mazón, J. N., Pardillo, J., & Trujillo, J. (2008, November). Modelling ETL processes of data warehouses with UML activity diagrams. In OTM Confederated International Conferences" On the Move to Meaningful Internet Systems" (pp. 44-53). Springer, Berlin, Heidelberg.

Navathe, S. B., & Ra, M. (1989, June). Vertical partitioning for database design: a graphical algorithm. In ACM Sigmod Record(Vol. 18, No. 2, pp. 440-450). ACM..

Navathe, S., Ceri, S., Wiederhold, G., & Dou, J. (1984). Vertical partitioning algorithms for database design. ACM Transactions on Database Systems (TODS), 9(4), 680-710.

Neil, C. G., De Vincenzi, M. E., & Pons, C. F. (2014). Design method for a Historical Data Warehouse, explicit valid time in multidimensional models. Ingeniare. Revista chilena de ingeniería, 22(2).

OMG-BPMN: Business Process Model and Notation (BPMN). Version 2.0.2 (December 2013).https://www.omg.org/spec/BPMN/2.0.2/

OMG-MDA: Model Driven Architecture (MDA) (2001) Document number ormsc/2001-07-01.

OMG-MDA: MDA Foundation Model (2010) OMG document ormsc/10-09-06

OMG-MDA: Model Driven Architecture (MDA) (2014) MDA Guide rev. 2.0 OMG Document ormsc/2014-06-01.

OMG-MOF: OMG Meta Object Facility (MOF) Core Specification (Novembre 2016) Version 2.5.1.

OMG-QVT: Object Management Group, MOF 2.0 (June 2016) Query/View/Transformation Specification. Version 1.3

Özsu, M. T., & Valduriez, P. (2011). Principles of distributed database systems. Springer Science & Business Media.

Pardillo, J., Mazón, J. N., & Trujillo, J. (2008, September). Model-driven metadata for OLAP cubes from the conceptual modelling of data warehouses. In International Conference on Data Warehousing and Knowledge Discovery (pp. 13-22). Springer, Berlin, Heidelberg.

Pardillo, J., & Mazón, J. N. (2011). Using ontologies for the design of data warehouses. arXiv preprint arXiv:1106.0304.

Petrović, M., Vučković, M., Turajlić, N., Babarogić, S., Aničić, N., & Marjanović, Z. (2017). Automating ETL processes using the domain-specific modeling approach. Information Systems and e-Business Management, 15(2), 425-460.

Pinet, F., & Schneider, M. (2009). A unified object constraint model for designing and implementing multidimensional systems. In Journal on Data Semantics XIII (pp. 37-71). Springer, Berlin, Heidelberg.

Ponniah, P. (2004). Data warehousing fundamentals: a comprehensive guide for IT professionals. John Wiley & Sons.

Prakash, D., & Prakash, N. (2018). Data Warehouse Requirements Engineering. Springer Singapore.

Prakash, N., & Gosain, A. (2008). An approach to engineering the requirements of data warehouses. Requirements Engineering, 13(1), 49-72.

Prakash, N., & Gosain, A. (2008). An approach to engineering the requirements of data warehouses. Requirements Engineering, 13(1), 49-72.

Prat, N., Akoka, J., & Comyn-Wattiau, I. (2006). A UML-based data warehouse design method. Decision support systems, 42(3), 1449-1473.

Romero, O., & Abelló, A. (2007, November). Automating multidimensional design from ontologies. In Proceedings of the ACM tenth international workshop on Data warehousing and OLAP (pp. 1-8). ACM.

Romero, O., & Abelló, A. (2010). A framework for multidimensional design of data warehouses from ontologies. Data & Knowledge Engineering, 69(11), 1138-1157.

Romero, O., & Abelló, A. (2011). Multidimensional design methods for data warehousing. In Integrations of Data Warehousing, Data Mining and Database Technologies: Innovative Approaches (pp. 78-105). IGI Global.

Saber, B. (2011). Open source comme système d'informatique décisionnelle. Revue internationale d'intelligence économique 2011/1 (Vol 3), p. 93-101

Saroop, S., & Kumar, M. (2011, June). Comparison of data warehouse design approaches from user requirement to conceptual model: A survey. In Communication Systems and Network Technologies (CSNT), 2011 International Conference on (pp. 308-312). IEEE.

SAS Institute Inc (2004) SAS 9.1 OLAP Server. MDX Guide. Cary, NC:SAS Institute Inc.

Shukla, A., Deshpande, P., Naughton, J. F., & Ramasamy, K. (1996, September). Storage estimation for multidimensional aggregates in the presence of hierarchies. In VLDB (Vol. 96, pp. 522-531).

Soler, E., Trujillo Mondéjar, J. C., Blanco Bueno, C., & Fernández-Medina Patón, E. (2009). Designing secure data warehouses by using MDA and QVT.

Song, I. Y., Khare, R., An, Y., Lee, S., Kim, S. P., Kim, J., & Moon, Y. S. (2008, October). Samstar: An automatic tool for generating star schemas from an entity-relationship

diagram. In International Conference on Conceptual Modeling (pp. 522-523). Springer, Berlin, Heidelberg.

Song, I. Y., Khare, R., & Dai, B. (2007, November). SAMSTAR: a semi-automated lexical method for generating star schemas from an entity-relationship diagram. In Proceedings of the ACM tenth international workshop on Data warehousing and OLAP(pp. 9-16). ACM.

Spofford, G., Harinath, S., Webb, C., Huang, D. H., & Civardi, F. (2006). MDX Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase. New York: Wiley.

Tannenbaum, A., & Foreword By-Simon, A. (2001). Metadata solutions: using metamodels, repositories, XML, and enterprise portals to generate information on demand. Addison-Wesley Longman Publishing Co.

Toumi,L. (2015) .Optimisation des performances par administration et tuning d'entrepôt de données

Toumi, L., Moussaoui, A., & Ugur, A. (2015, November). EMeD-Part: An Efficient Methodology for Horizontal Partitioning in Data Warehouses. In Proceedings of the International Conference on Intelligent Information Processing, Security and Advanced Communication (p. 43). ACM.

TPC-DS : Transaction Processing Performance Council (TPC) (2018). TPC Benchmark DS - Standard Specification, Version 2.9.0 http://www.tpc.org/tpcds

Trujillo, J., & Luján-Mora, S. (2003, October). A UML based approach for modeling ETL processes in data warehouses. In International Conference on Conceptual Modeling (pp. 307-320). Springer, Berlin, Heidelberg.

Vaisman A, Zimányi E (2014) Data Warehouse Systems Design and Implementation. Springer.

Vassiliadis, P., Simitsis, A., Georgantas, P., Terrovitis, M., & Skiadopoulos, S. (2005). A generic and customizable framework for the design of ETL scenarios. Information Systems, 30(7), 492-525.

XML for analysis (XMLA), https://msdn.microsoft.com/en-us/library/ms186604.aspx

Zepeda, L., Ceceña, E., Quintero, R., Zatarain, R., Vega, L., Mora, Z., & Clemente, G. G. (2010, March). A MDA tool for data warehouse. In Computational Science and Its Applications (ICCSA), 2010 International Conference on (pp. 261-265). IEEE.

Zepeda, L., Celma, M., & Zatarain, R. (2008, June). A mixed approach for data warehouse conceptual design with MDA. In International Conference on Computational Science and Its Applications (pp. 1204-1217). Springer, Berlin, Heidelberg.

Zepeda, L., Santillan, L., Cecena, E., Manjarrez, E., Vega, L., & Garcia, C. (2015, September). A meta-model based approach for Data Warehouses design and implementation. In Database and Expert Systems Applications (DEXA), 2015 26th International Workshop on (pp. 212-216). IEEE.

Zhang, Y., & Orlowska, M. E. (1994). On fragmentation approaches for distributed database design. Information Sciences-Applications, 1(3), 117-132.

Ziyati ,E. (2010) Optimisation de requêtes OLAP en Entrepôts de Données Approche basée sur la fragmentation génétique